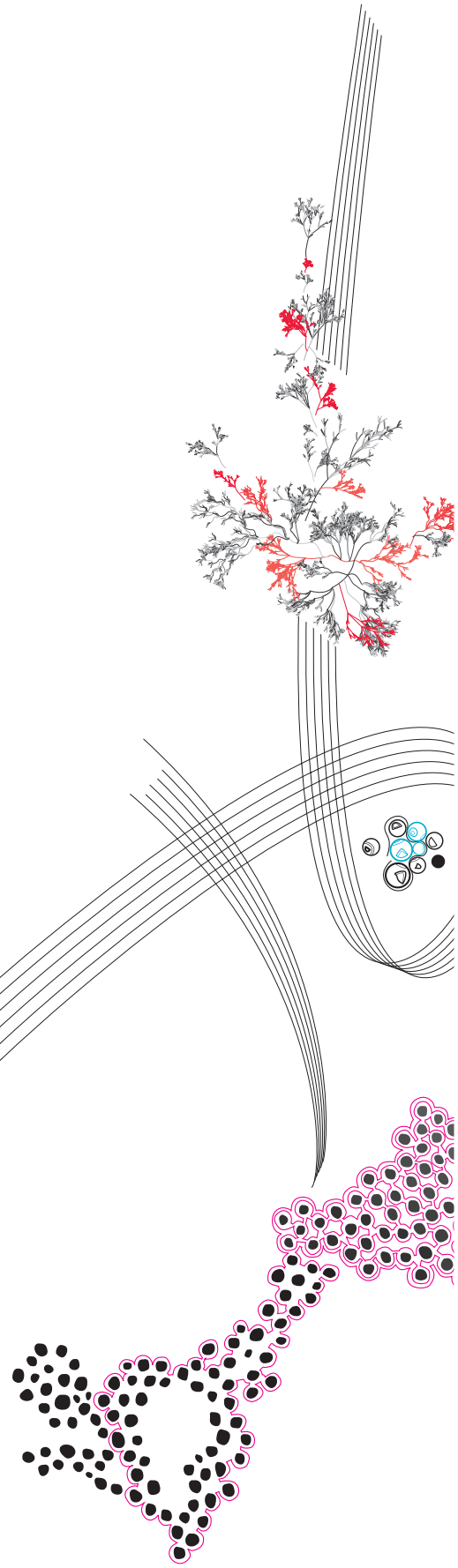


Master thesis
Industrial Engineering & Management

# Designing a complex order acceptance tool at Limis B.V.

Nick Sligting

University Supervisor:
Dr. Engin Topan
Dr. Ipek Seyran Topan

Company Supervisor:
Hans ten Brug
Jur ten Brug

January 25, 2024

Faculty of Behavioural,
Management and Social Sciences
University of Twente

UNIVERSITY OF TWENTE.

# Executive Summary

This study was conducted at Limis, which is the developer of advanced planning and scheduling (APS) software called Limis Planner also referred to as Limis Start. Limis users, typically operate job shop-like manufacturing systems in which they make specialized (partial) end products in response to customer demand. Therefore, they need to respond to dynamically arriving requests containing one or multiple orders consisting of one or multiple products. Based on available resource capacity (machines, materials, etc.), they need to determine whether they can and want to deliver/ fulfill the order by the requested due date. The order acceptance module of Limis Planner helps its users make this decision. It currently does so by using Limis Planner's built in regular scheduling function, which requires a lot of computational effort and thus results in a high computation time. For the larger clients especially, this causes a high customer response time. This study aims to design a fast and reliable order acceptance method and develop a tool that enables all Limis customers to respond to dynamically arriving customer orders in mere minutes.

In designing a suitable solution to the order acceptance problem, we had to take into account that their customers typically work with orders with complex assembly-type routing, various machine groups with multiple (identical), and possibly even external, resources, with varying day specific operating hours. Furthermore, we learned that the scheduler generates schedules based on dispatching rules, which use release and due date derived from an infinite backward loading procedure.

Extensive literature research revealed the assumptions and limitations of studies conducted to address similar problems. Order acceptance typically concerns wanting to reserve capacity for potentially more rewarding orders that could arrive in the future. This is however, of no concern for us. We instead want to figure out whether or not we can accept an order given a certain due date. If this is not the case, we want to either know what the earliest possible delivery date is, or where we need to increase capacity so that we can adhere to the requested due date. As a side objective we also explored AI / reinforcement learning based solutions, but came to the conclusion that these generally do not fit our context.

Taking into account our objectives, we propose A finite loading based method in combination with an iterative approach. To initially check whether an order fits before a due date we employ backward finite loading, as this allows us to work backwards from the requested due date and cut off the procedure when we pass the current date. For the capacity adjustment part we combine this with an iterative approach that decreases the processing time of orders over time, to mimic an increase in capacity. We do this to find a minimal increase in capacity that allows the order to fit before a due date. For the due date assignment part we instead use forward finite loading, as we want to find the earliest possible date on which we are able to deliver.

For the ("must-have") iterative capacity adjustment procedure we experiment with the size of the processing time decrease per iteration, as well as the method for selecting which operation to adjust next. These numerical experiments have shown that it should be ran with a step size of 5 minutes, and should generally use the Highest Order Level (HOL) method to select the next operation to adjust, as these lead to the best results on average.

When it comes to performance we can say that the proposed new solution is many orders of magnitude quicker than the current method. In fact, for most clients it takes but a faction of a second, compared to on average 20 to 30 minutes. Especially for the larger customers, for whom the old method takes several hours, this is a almost unimaginable improvement.

When it comes to the accuracy/ the degree of correctness of the proposed tool, we found that the results are incomparable to those of the old method and scheduler at this time. The biggest reason for this is that we were not able to work with machine capacities on an individual level, but had to make due with capacity per machine group, unlike the scheduler. This because outside of the scheduling procedure, Limis planner does not store machine capacities on an individual machine level. Therefore, the results are drastically different at this point of time.

Instead of cherry picking cases in order to be able to pretend the proposed method is highly accurate, we chose to leave further evaluation of the solution quality outside of the scope of this study. This because we believe this would have strongly hurt the integrity and validity of this study. We have however shown that the method/ tool produce valid results, i.e., it validly loads each operation of an order under the assumption of pooled machine capacity per machine group.

Sadly it was not feasible to adjust Limis planner to work with machine capacities on individual machine level within the remaining time frame of this study. The exact performance of the tool, in regard to being able to replicate results found by the scheduler will have to remain to be seen. None the less, the method is very promising and assuming Limis makes the required adjustments in the future, we find that proposed solution is far superior to the current method. However, we do believe that both methods can exist next to each other, in which the old method can play a role when one wants to know the detailed scheduling outcome, and computation time plays no role.

Next to designing the proposed order acceptance method, we have also developed a tool capable of applying the new method, to real customer data straight from their database. This solution is fully scale able and should work for any customer, regardless of their manufacturing environment. Furthermore, we've made inroads to integrating this tool into existing software, allowing it to work with the interface of the existing order acceptance module within Limis Planner.

Our recommendations to Limis are therefore, to finish developing the proposed solution. To that extend we suggest expanding on the developed tool and fully integrating this within Limis Planner. Again the most important thing would be to first enable the use of capacity levels on individual machine basis. Thereafter, we would strongly urge Limis to conduct thorough experimentation on the accuracy of the proposed method's found results.

From thereon, we would recommend having an actual customer test the tool further, after which the tool could be further pushed to a sort of beta release for all interested customers.

Furthermore, we found that there is no real relevance, and certainly no need, for Limis to use AI based methods in order acceptance at this time. On one side you could argue that, considering the limited data available for training, the results of the classification models look promising. On the other side however, even the results of the best performing model, the decision tree, are not all that great, especially looking at the results of the 5-fold cross validation. These results indicated only a 75% accuracy, meaning one in four cases was classified incorrectly. Furthermore the low precision particularly seems to be an issue as the results indicate that we would roughly, incorrectly classify 15% as fitting, while they do in fact not fit, resulting in a lot of scheduling problems down the line and in turn leading to unsatisfied customers.

The further reduction in computation time that classification models could realize over the proposed finite loading method, is really not required for any imaginable practical scenario as the proposed method is more than quick enough. Furthermore, we are of strong believe that once allowed to work with individual machine capacities and fully developed, the proposed finite loading based method will be more than accurate enough for Limis' customer's business needs. Whereas we doubt that the classification model method will ever be able to achieve this.

We therefore, do not recommend wasting time and resources on developing a classification model based approach at the current moment in time. Perhaps this could be revisited in the future if Limis has spare time and/or resources.

We do believe that the developed tool itself, assuming earlier mentioned updates to the underlying method, is close to being ready for a beta release to customers. With a little additional effort it can fulfill all their customer's order acceptance related needs. This notion is further confirmed, by the positive feedback received after showcasing the tool at an actual Limis client.

# Acknowledgements

The completion of this thesis marks the end of my master's program at the University of Twente and thereby the end of my time as a student. This is a milestone I've been working towards for several years. I would like to thanks some of the people without whom this would not have been possible.

First of all, I would like to thank my University supervisors Dr. Engin Topan and Dr. Ipek Seyran Topan, whose feedback and guidance was simply instrumental to the success of this thesis.

Secondly, I would like to thank my colleagues at Limis, and particularly my company supervisors Jur and Hans ten Brug for their guidance, support, and giving me the opportunity to conduct my thesis within their company.

Lastly, I would like to thank my friends and family for their everlasting support throughout the years, without which I would not be presenting this thesis today. Thank you for always being there for me.

I hope this work sparks further exploration and contributes meaningfully to future operations at Limis.

Nick Sligting

# Contents

# Glossary

**(C)FFL** (Cummulative) Finite forward loading.

**AI** Artifical intelligence.

**APS** Advanced planning and scheduling.

**Critical path (length)** The shortest amount of time in which an entire order can be finished.

**DDA** Due date assignment.

**ML** Machine learning.

**MRP** Material requirements planning.

**OA** Order Acceptance.

**RL** Reinforcement learning.

# Chapter 1

# Introduction

The purpose of this master thesis is to advice Limis, an APS software developer, on how to employ reinforcement learning to improve and speed up their order acceptance tool. The order acceptance tool is an important feature of Limis Planner. It helps their customers, which are mostly manufacturers, make decisions on whether to accept or reject new order and determines their possible delivery dates. This chapter discusses the organizational context, the research motivation, the problem statement, the research goal, the research approach and finally the research scope.

## 1.1 Organizational context and research motivation

In this first section we discuss the organizational context and research motivation. We start by taking a closer look at Limis. We then briefly discuss order acceptance, and finally we go into the research motivation.

### 1.1.1 About Limis

Limis B.V. is the software developer of Limis Planner, which is an Advanced Planning and Scheduling (APS) tool. Limis Planner is a software solution for production planning at manufacturing companies. The customers are typically in discrete manufacturing having job shop production system characterized by a large variety of products producing in small batches. Limis Planner is connected to the ERP systems of manufacturers extracting the data about production orders, products, order status, stocks and producing schedules and feeds that to ERP system. Limis Planner calculates, simulates, and visualizes the optimal production planning, taking into account finite and infinite capacities such as people, machines, tools, raw materials, purchases and subcontractors. Limis Planner has a modular structure, which allows to design the software to the needs, sizes, and complexities of different companies, e.g., scheduling function, machine, material, workforce, and tool planning.

At users of Limis planner, customers may request multiple products, leading to multiple orders. Orders often consists of multiple jobs/ operations, that each may require specific machines, materials and employees. These jobs may even consist of operations at an external subcontractor. When an order arrives, and if it is not possible to fill the order in its entirety before the due date, there are several alternatives to consider. First, they can try to negotiate a later due date, at which the order is to be delivered in full. Second, assuming that it is an order consisting of more than a single unit, partial delivery might

be an option. In this case as many as possible unit are delivered upon the requested due date, and the rest of the order is delivered at a later date. In case of an important order, the company might decide to either delay other orders that were previously planned, or if possible to work overtime and/or employ subcontractors.
All things considered, there are three types of flexibility that can be considered in accepting a new order that can't be filled in its entirety before the requested due date as is:

1. **Due date flexibility** Delivering at an alternative later date.
2. **Capacity flexibility** Reassigning employees, working overtime, and/or employing subcontractors.
3. **Quantity flexibility** Partial delivery before due date, and full order fulfilment at a later date.

Each of these types come with their own optimal approach and things to consider, e.g. capacity flexibility might be possible if the customer is willing to pay for overtime. Whether or not you want to accept an order under these conditions falls under the domain of order acceptance, as will be further explained later on.

### 1.1.2 Order acceptance

The Problem of order acceptance and scheduling (OAS), or here simply order acceptance (OA), is defined as the joint decision of which orders to accept for processing and how to schedule them. Slotnick (2011) describes it as the problem of dealing with a collection or stream of orders whose combined processing requirements would exceed available capacity, in which some orders can be rejected. If no orders can be rejected the problem reduces to that of scheduling. If no scheduling is required, the problem is equal to the knapsack problem. The term order acceptance and its abbreviation OA are used interchangeably throughout this report. However, as has become apparent after discussion with executives at Limis, in practice the order acceptation decision is always to accept, i.e., it's much more a question of how can we make this order work and by when can it be delivered. Therefore, what started out as a project about order acceptance quickly turned into a project on the topic of rescheduling and capacity planning, in which the main concern is finding possible bottlenecks preventing the acceptance of an order or an alternative delivery date.

One of the functionalities of Limis Planner is order acceptance (OA). With the order acceptance module manufacturers are shown the impact of accepting new orders on existing planning and production. It calculates an expected delivery date for the new order or, if given a specific delivery date, it can simulate the impact on/ conflicts with existing orders, e.g., what orders will be delayed. At least that is the idea behind it, what it does in practice is run the scheduling procedure again to create a new secondary schedule, i.e., it doesn't affect the actual current live schedule, and only exists within the order acceptance module. It then returns the found date for the order under evaluation. If this is earlier than the requested due date, the order can be delivered on or before the requested due date and thus can be accepted as is. Notably the addition of the order(s) under evaluation can completely change the previously existing schedule. This because the scheduler is run from scratch, leaving only the work currently in progress in tact.

### 1.1.3 Research motivation

Ideally the order acceptance process is quick enough to be able to make a decision and/ or convey a delivery date on the same initial phone call as the customer placing their order. Currently however, this is not at all the case. For any decent sized company, which can easily have thousands of scheduled orders, this can take up to several hours or in extreme cases even longer. An example of this is a customer that had more than a 50,000 operations that needed to be scheduled taking into account materials, machines and, employees, in which the scheduler took nearly 6 hours to complete.

## 1.2 Problem Identification

To investigate how we can reduce the order acceptance computation time, we create a problem cluster to identify the cause-and-effect relationships that lead to the core problem(s). To be able to create the problem cluster we first, in consultation with the organization, inventory all the related problems. Figure 1.1 shows the resulting problem cluster.

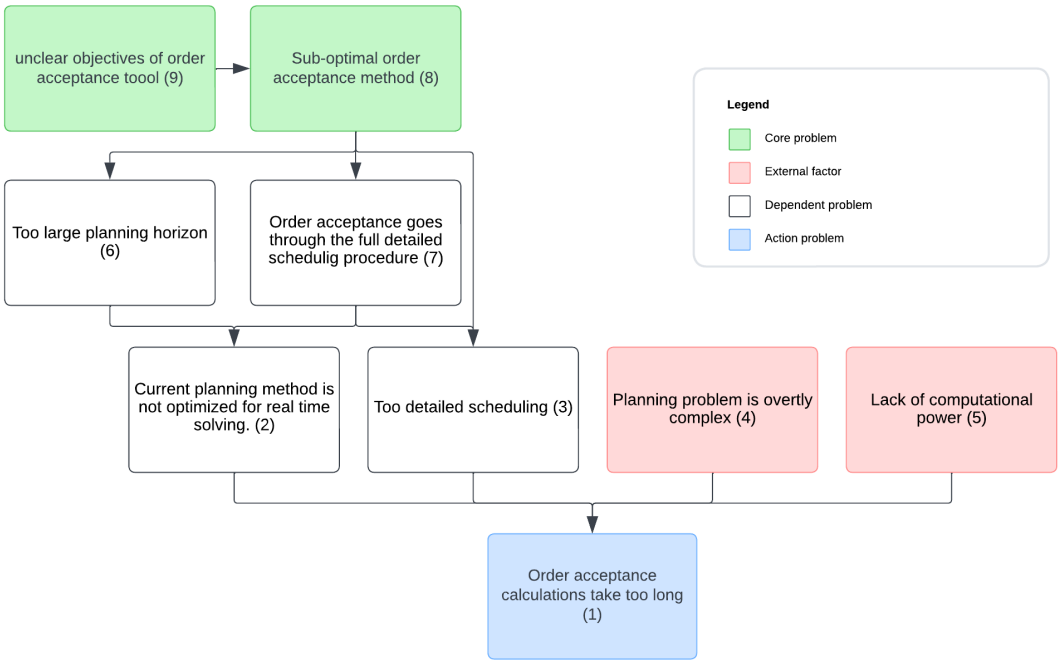



FIGURE 1.1: Problem cluster

The following problems are identified:

**(1) computation time is too long:** it is currently not possible to run the order acceptance calculations in a short enough time span to have the results ready within the same phone call in which an order is placed.

**(2) scheduling method not optimized for real time solving:** The scheduling method employed by the scheduler was not designed for/ with real-time solving as first thought. It could very well be that there are other, faster scheduling algorithms available that do the job just as well from a order acceptance standpoint.

**(3) Too detailed scheduling:** Currently all planning involved in order acceptance decisions is done with great detail. E.g. a job is scheduled to be finished on a specific date and time, say Tuesday 18 May by 10:06. This level of detail is however not necessary for quick initial order acceptance decisions. Being able to know roughly the day of the week or in some cases just the week is enough. Calculating things this precise therefore can take a lot of unnecessary computation time.

**(4) Planning problem is overtly complex:** Some situations at customers might result in overtly complex scheduling problems, e.g. tens of thousands of operations need to be scheduled over 100 workplaces/ machines. Naturally the more complex and/or the larger the solution space of a planning problem is, the longer the computation time will be. The reality is however that these type of customers do exist and a solution has to work for these cases as well. Therefore we also consider this an external facor/ consider it to be out of our hands, I.e. we can not influence this.

**(5) Lack of computational power:** More computational power almost always result in less computation time. In that sense you could of course always try to reduce the computation time needed by simply increasing your computational power to seemingly infinite. This is however both outside of the scope of this research as well as infeasible from a practical standpoint.

**(6) Too large planning horizon:** Currently the order acceptance tool has to recalculate/ go through all planned operations, i.e., the entire outstanding schedule. From the aspect of the one job one has the order acceptance problem for this is a way too large planning horizon, e.g., if we have a small job that we want to fit in, ideally next week, we shouldn't need to recalculate all scheduled operations months from now.

**(7) Order acceptance is done through the full scheduler:** Currently the order acceptance decisions are calculated by the full detail scheduler, that is used in the planning and shop floor control tools.

**(8) OA method does not fit objectives:** Currently the order acceptance uses the full detail scheduler (7) and goes through all currently scheduled operations (6). Doing so it calculates everything in great detail, accurate to the minute.(3) Seemingly this does not align with the objective of solving the order acceptance problem initially.

**(9) Unclear objectives of the OA tool:** It is unclear what exactly Limis wants to achieve with their OA tool, which of course leads to the use of a sub-optimal method (8). Only when we know what we want to achieve, can we find an optimal method to do so.

### 1.2.1 Core problem

Now that the involved problems and their relations are clear, the core problem can be selected. Heerkens & van Winden (2012) state that to find the core problem you must look at the problems in the problem cluster that are the furthest away and thus have no causes themselves. We quickly see that in our case, the core problem are the unclear objectives of the OA tool and the thereby resulting sub-optimal method employed by the OA tool. We treat these 2 problems together as the core problem, as they go hand in hand. The current planning method is purely rule-based and for the most part does not take in account stochasticity. I.e., does not consider the stochastic nature of process times and resource availability. Furthermore, the current employed method tries to work

out everything exactly, resulting in exact delivery dates. While this is great for a lot of planning scenarios on a day-to-day basis, this level of detail is often not needed in the context of OA decisions.

### 1.2.2 Action problem

The action problem can be formulated as follows:

**"Order acceptance calculations can currently take up to multiple hours compute, while Limis wants them to take at most 2 minutes."**

Clearly the norm is for order acceptation computations to take at most 2 minutes, whereas the reality is currently that they may take up to several hours. While the norm is rather strict, the reality is inconsistent as the computation time is highly dependent on the complexity of the case. Nevertheless, the discrepancy between norm and reality is seemingly rather large.

### 1.2.3 Research problem

According to the rules of Heerkens & van Winden (2017) the problem identification brings us to the following research problem:

**The current method used by the order acceptance tool at Limis results in too long computation times.**

A new method/ technique for making order acceptance decisions should help overcome this problem. This research investigates (AI based) capacity loading and rescheduling techniques and advises on how to employ these to improve "order acceptance" calculations so that the computation time can drastically be decreased. At the basis of doing so is first clearly defining the desired objectives/ goals of the order acceptance tool, as only after these are clear can an appropriate method be formulated.

## 1.3 Research goal

In this section the research goal/ objectives and research questions are formulated.

### 1.3.1 Main research goal

The organization aspires to have an order acceptance tool that can perform initial order acceptance decisions in mere minutes, i.e., it has to be quick enough to be able to decide and/ or propose a delivery date during the same call as the customer placing an order. The goal of this research is therefore, to develop a tool that can solve customer order acceptance problems in a short time span. This research goal translated to the following main research question:

**How can a quick order acceptance tool be designed within Limis Planner in order to respond to, and act on arriving customer orders?**

To achieve this research goal and answer the main research question we draw up additional research questions as show in section 1.3.2.

### 1.3.2 Research questions

The following research questions combined serve to answer the main research question. They are formulated so that each research question and its sub questions correspond to a chapter. The approach to solving the research questions is explained in section 1.4.

1. What does Limis want to achieve with their order acceptance tool? (chapter 2)
    - What is the desired functionality of the order acceptance tool within Limis Planner?
    - What does the current order acceptance tool do and how does it work?
    - What data is available for making order acceptance decisions?
2. Which methods for order acceptance exist in literature? (chapter 3)
    - What does the order acceptance process look like under various production environments?
    - What methods exist in literature that can be used to solve or aid in order acceptance problems?
    - Which AI based approaches exist in literature related to these methods?
3. How can we develop the desired order acceptance tool? (chapter 4)
    - What methods can be used to perform each of the desired aspects of the order acceptance tool?
    - What are the outcomes of the proposed solution method?
    - How does our proposed tool perform in comparison to the current order acceptance tool?
    - How can the proposed tool be used to solve order acceptance problems?
    - How can the proposed tool be integrated into existing software?
4. How can we validate the proposed solution method and its resulting tool? (chapter 5)
5. How does our proposed method perform? (Chapter 6)
    - How can we test the performance of our proposed method?
    - How does our proposed tool perform in comparison to the current order acceptance tool?
    - How does the alternative classification model based method perform?

## 1.4 Research approach

As indicated by the research questions the report continues in the following structure. Chapter 2 describes the current situation at Limis with regards to order acceptance as well as the desired functionalities of the order acceptance tool. Chapter 3 proceeds with a literature review that is aimed at describing order acceptance problems, finding methods to deal with the order acceptance problem at hand and, finding possible AI based (reinforcement learning) approaches to these methods. Chapter 4 then describes the proposed solution in great detail, the assumptions and simplifications, and how to use the actual tool itself. Chapter 5 follows up with an extensive look at the validation of the proposed solution methods and the proposed tool. Chapter 6 describes the performed numerical experiments and their results, including the performance of the proposed method. Finally chapter 7 contains the discussion of the results, the research limitations, the scientific contributions, recommendations to the organization, and recommendations for future research.

### 1.4.1 Research design

In order to acquire the knowledge necessary to solve the problem that we are facing, rigorous literature research into order acceptance problems and methods to solve them will be performed. Furthermore, we will also specifically look at machine- and reinforcement learning in combination with scheduling and order acceptance problems. The knowledge gained therefrom will be used to propose, build, and deploy a practical tool to solve OA problems. This in an effort to eventually create the tool necessary to solve our problem. In doing so we will require a lot of data to build the model, data such as machine processing times, resource availability, etc. This data is to be obtained from existing data stored in Limis Databases. Using SQL, we will extract the data after which we will clean it and use it as a source for our model variables.

### 1.4.2 Choices

#### Solution quality vs computation time

It is likely that we will have a trade-off, at least to some extent, between accuracy (solution quality) and computational speed. Fortunately, in our case we can afford to lose some accuracy in the OA problem. That is, we do not need to know the exact date and time the order can be delivered, but rather we just want to know roughly what week it can be finished by. This means that we deliberately choose for a less accurate but therefore also less computationally taxing solution.

### 1.4.3 Scope

The scope of the research is limited to order acceptance and solutions that do not require overhauling the complete detailed scheduling process itself. Due to time restrictions we are purely interested in creating a fast order acceptance tool, not in full detailed scheduling. Furthermore, we disregard stochasticity in things like processing times and machine failures. Note however, that the scheduler in Limis planner creates schedules with a certain robustness to them to account for some of this stochasticity, e.g., it adds fixed waiting times after each operation. Finally, we are not keeping in mind potential future orders in order acceptance decisions, i.e., we're not reserving space in the schedule for orders that might or might not arrive.

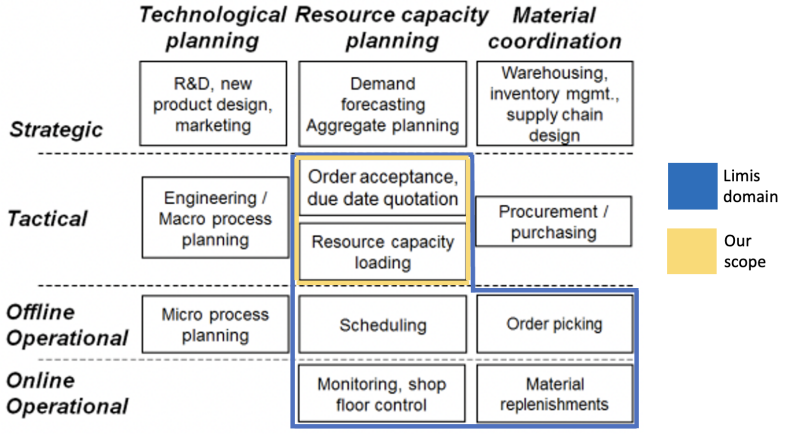



FIGURE 1.2: Our problem/ scope within the planning positioning framework

# Chapter 2

# Problem context

In this chapter we dive deeper into the context of the problem, i.e., we discuss various aspects of the situation at hand. We start by discussing the data that is available to us at Limis. We then move on to how the underlying scheduler works as well as the tracking and graphing of machine capacities. Finally we discuss the current order acceptance tool and its use within Limis planner.

## 2.1 Input data

This section concerns the data that is available at Limis/ is used in Limis Planner regarding customer's production resources and production orders.

### 2.1.1 Production resources

Production resource data includes essential information regarding workstations, employees, materials, and tools. A workstation can be a single machine or a group of identical machines. Employee information includes their regular daily working hours and the workstations that they can be assigned to work at. For each workstation that an employee can be assigned to, he or she may be involved in setup, processing, or both. This is accompanied by a skill level ranging from 0 to 9. Skill levels are used during scheduling to ensure the most suitable employees are assigned to each of the workstations.

While workstations are theoretically always available, their operability is constrained by employee availability, i.e., machines can also only operate during working hours. Some operations may require multiple employees while others can be completed by a single employee but can be completed faster if multiple employees are assigned to the job. The number of employees assigned to a job is denoted by the crew factor. Some jobs/ operations are outsourced to external organizations. In this case the machine required for the operation is registered in Limis Planner as an external workstation.

Materials and their availability are taken care off by the build in MRP functionality in Limis Planner. During scheduling the on hand inventory is determined after each step/ decision based on starting inventory levels and material lead times. Of course this depends on whether the customer chooses to use the MRP functionality, i.e., not all customers include material requirements in their planning. This could be for various reasons, like the customer always having plenty of materials in stock or only performing operations that require no additional materials. So for the customers that do include materials in their

planning, we know the lead times for each of their required materials.

These production resources shape the manufacturing environment and thereby define the constraints of the order acceptance (and scheduling) problem. They are all static, known values and are not decision variables in our problem.

### 2.1.2 Production orders

Production orders are categorized into two types: make-to-stock orders (stock orders) and make-to-order orders (customer orders). They consist of operations that require specific workstations, tools and possibly employees. Operations typically cannot be interrupted and may involve setup time and waiting time before/ after processing. A production order can follow either a linear routing, where each operation has at most one predecessor and at most one successor, or an assembly-type routing, where operations can have multiple predecessors.

In most cases the due dates and quantities of customer orders are specified by customers themselves. In other cases, the due dates and quantities of stock orders are determined by the internal or external Materials Requirements Planning (MRP) function. Additionally, the production specifications (routing, processing times, and material requirements) of a make-to-stock order can be derived from the bill of materials and the routing of the corresponding make-to-stock item. However, customer orders for make-to-order items often have unique specifications, such as personalizations like specifically requested designs or dimensions.

Each order and its tasks are assigned a priority rating ranging from 0 to 9, which reflects its level of priority, i.e., a job with rating 5 has priority over an operation with rating 4 and below. These priorities are used in detailed scheduling by means of a dispatching rule, as described in Section 2.2.2.

## 2.2 Scheduling

As is common for most manufacturers, users of Limis planner operate in a dynamic environment, e.g. (priority) orders arrive, machines break down, employees get sick, etc. Limis planner constantly reschedules to respond to these dynamically changing conditions. If for example an employee, that is the only one that can operate a certain machine, can't come in to work, it is important to update the schedule promptly. This allows them to minimize time waste resulting from the machine/ workstation downtime. Incoming orders that are not high priority and/or in a rush, that arrive continuously over time, are incorporated in the schedule periodically, mostly on a daily basis.

To take into account stochasticity in processing, waiting and setup times, Limis planner has a functionality in which a job's processing time estimate can be increase by a factor. Essentially Limis planner treats the scheduling problem in static deterministic manner. Every time the scheduler is run, the material requirements planning (MRP) is automatically ran beforehand. This provides the scheduler with essential input such as on-hand inventory of items/ material and their upcoming delivery dates and quantities.

The scheduling in Limis planner consists of two phases, rough planning and detailed

scheduling. The scheduler first makes a rough planning based on due dates and processing times that then serve as input for the detailed planning. The most notable difference between the two phases is that, unlike in detailed scheduling, the rough planning does not take into account any resource limitations. That means that all material, man, and machine capacities are assumed to be infinite. The full scheduler is usually ran at least once a day, often during the night. Figure 2.1 shows the relationships between the two phases and the various outputs of Limis planner. In the next two sub sections both phases of the scheduler are discussed in more detail.

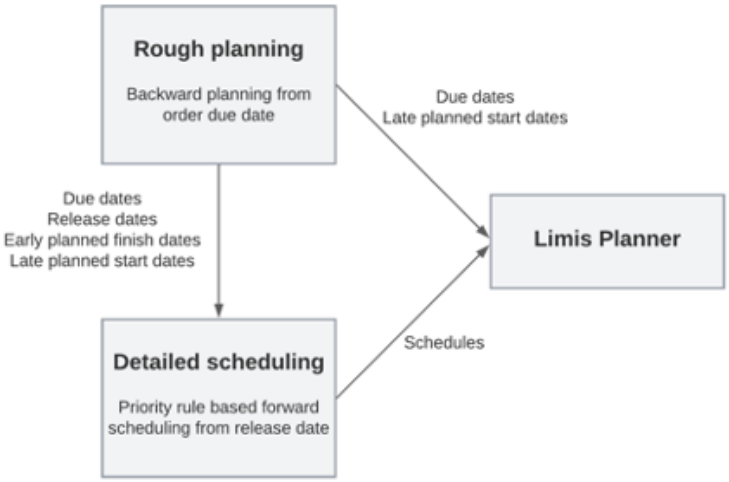



FIGURE 2.1: Structure of Limis Planner

### 2.2.1 Rough planning

The rough planning in Limis planner employs a backward infinite loading method and takes into account neither existing workload at workstations nor material/ component availability. In the rough planning every task is planned backwards from its due date. This results in both a planned- and a critical start date on both order- and task level. Figure 2.2 depicts the backward planning of an order and the operations/ tasks it consists of. All tasks are assigned a start date and the sum of all operation processing times belonging to an order is the throughput time of that customer order. Note that each job includes a small waiting time to account for possible delay before a job start (due to possible lack of resource availability). This waiting time is specified when first setting up Limis planner in consultation with Limis consultants. An often used waiting time is 5 minutes.

The shortest possible amount time in which the entire order can be finished is called the (optimal) critical path. The critical start date is the due date minus the critical path. If the critical start date is exceeded the order will always be late, unless intervened by working overtime and/ or subcontracting certain operations. The due date of a task is equal to the critical start date of its successor while the due date of the final task is equal to the order due date.

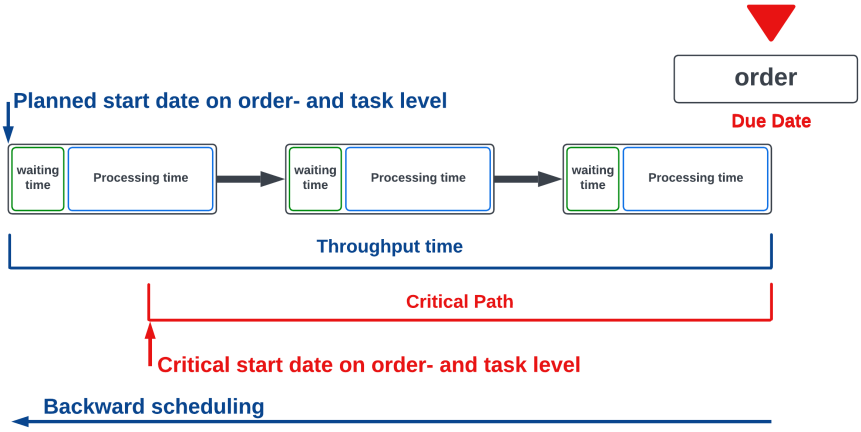



FIGURE 2.2: Planning of an order and all its operations

The throughput time of a task is calculated using equation 2.1. This includes waiting time, setup time, processing time, and possible waiting- (after processing) and transportation time. These are fixed known inputs, only the machine capacity is date specific.

$$\tau = W + S + P/(MC \times CF) + Q + T \tag{2.1}$$

$\tau$ = throughput time of a task, in days.

**W** = Waiting time before processing in days.

**S** = Setup time in days.

**P** = Processing time in hours.

**MC** Daily machine capacity in hours.

**CF** = Crew factor (number of employees that can work simultaneously)

**Q** = waiting time after processing in days.

**T** = Transportation time in days.

Crew factor (CF) in equation 2.1 is only ever not 1 in case of a situation in which multiple employees can work simultaneously on a task and thereby speed up processing time, e.g., a job takes 10 hours for a single employee but takes only 5 hours if two employees can both perform half of the job simultaneously. Setup time and post processing waiting can depend on the workstation or may be job/ order specific. Transportation time of the final task is not considered. The waiting time before processing is an estimated by the company based on their experience of the shop floor and its employees.

The release date, the time at which a task becomes available to plan, is based on the planned finishing date of its predecessor. This due to the precedence constraint of the tasks within an order, i.e., a task can simply not start before its predecessor has finished.

The result of this rough planning phase is that every order is assigned a release and due date and that all of each orders' tasks are assigned a release date, due date, early planned finish date and late planned start date. These dates then server as input for the detailed scheduling phase. Optionally the scheduler does allow for delaying and order up until a number of days before its due date under the Just-in-time (JIT) principal. This is a setting that can be turned on if so desired.

### 2.2.2 Detailed scheduling

Unlike the rough planning, the detailed scheduling works forwards. It takes the output of the previous rough planning phase and determines definitive start and finish date of tasks and thereby also the full order start and finish times. The digital plan board in Limis Planner shows the visualized outcome of the detailed scheduling as Gantt charts for each machine/ work station. Furthermore all the tasks/ operations are listed in the to-do lists for each machine, work station and employee.

In the detailed scheduling material availability, as required by a task, is taken into account. in case of a lack of material the start/ release date of the task is delayed until its first (expected) availability. If there is no delivery planned for the required material within its lead time, the release date of the order is delayed by the required materials lead time.

The detailed scheduling phase is based on priority rules and results in a so called non-delay schedule in which no resource is kept idle if there is a possible operation that can be scheduled on that resource. This does mean that situations can occur in which the schedule suggest working on operations/tasks of which the due date is still (relatively) far away. In cases where this is not desired it is possible to give orders or its tasks release dates, meaning that they will not be scheduled before these dates. Figure 2.3 shows the flowchart of a simplified version of the detailed scheduling procedure.

At the start of the process the current moment in time is set as t = 0 and the 4 possible sets tasks can belong to are initialized. The active set contains the operations that are currently in progress, the finished set contains the operations that have finished processing, the decision set contains all the operations that have not yet been scheduled and that can start (because their predecessors have finished), and finally the waiting set contains all the other operations that are not in the other sets. These sets are used to distinguish between job status, so that the planner doesn't have to go over all the job all the time, speeding up the process. operations of which the predecessor hasn't finished yet can not start yet and therefore don't need to be considered for example. Of course operations that have already been finished also don't have to be scheduled again. When moving operations that can start at the current time t in step 1, it also takes into account resource availability (material, machine, operator, tools). This further reduces the set of operations that can be selected to be scheduled next. Table 2.1 shows most of the priority rules that are available to schedule tasks by, in Limis Planner. In the settings one can choose which (combination) of these rules should be applied based on the users desires. This is often determined upfront when first starting to use Limis Planner in consultation with Limis consultants.

Over time additional features have been developed and integrated into the detailed scheduling, based on situations encountered at customers:

Table 2.1: Common priority rules available in Limis Planner

| **Rule** | **Purpose** |
|---|---|
| Priority (PRIO) | This rule ranks orders based on their preassigned priority numbers. An order with a higher priority number takes precedence over an order with a lower priority number. This rule can be used to ensure that important or rush orders are handled first. |
| Earliest due date (EDD) | this rule ranks the operations based on their order due dates. The idea being to finish the the work that needs to be done the earliest first. |
| Earliest due week (EDW) | Same ass earliest due date except on week level. |
| Earliest operation due date (ODD) | this rule ranks the operations based on their order due dates. The idea being to finish the the work that needs to be done the earliest first. |
| Earliest operation due week (ODW) | same as earliest operation due date except on week level. |
| Earliest operation release date (ORD) | This rule ranks the operations based on the their release date. The idea of this rule is to minimize the variation in the waiting times. |
| Earliest operation release week (ORW) | Similar to ORD except on week level. The idea is that this one might perform better in certain scenarios in combination with job clustering. |
| Earliest operation release month (ORM) | Similar to ORD and ORW except on Month level. |
| Shortest processing time (SPT) | This rule sorts operations in ascending order of processing time. This aims to minimize queue sizes and thereby reduce average order flow time. |
| Shortest remaining processing time (RPT) | This rule sorts operations in ascending order of remaining processing time. The difference with SPT is that it takes into account the total number of hours required to finish all operations belonging to this order rather than just the processing time of the operation itself. |
| Longest remaining processing time (LRPT) | This rule sorts operations in descending order of remaining processing time. this is the exact opposite of RPT. |
| Shortest setup time (SST) | This rule sorts operations in ascending order of setup time. Mostly used in cases of sequence-dependent setup times and often used in conjunction with other rules such as ORW. Requires the user to enter (sequence-dependent) setup times for each operation. |
| Maximum lateness (ML) | This rule sorts operations based on their late planned start date in descending order. This aims to minimize the maximum lateness of the orders. |
| On-time (OT) | This rule sorts operations based on their late planned start date in ascending order. this is the exact opposite of the maximum lateness rule. |

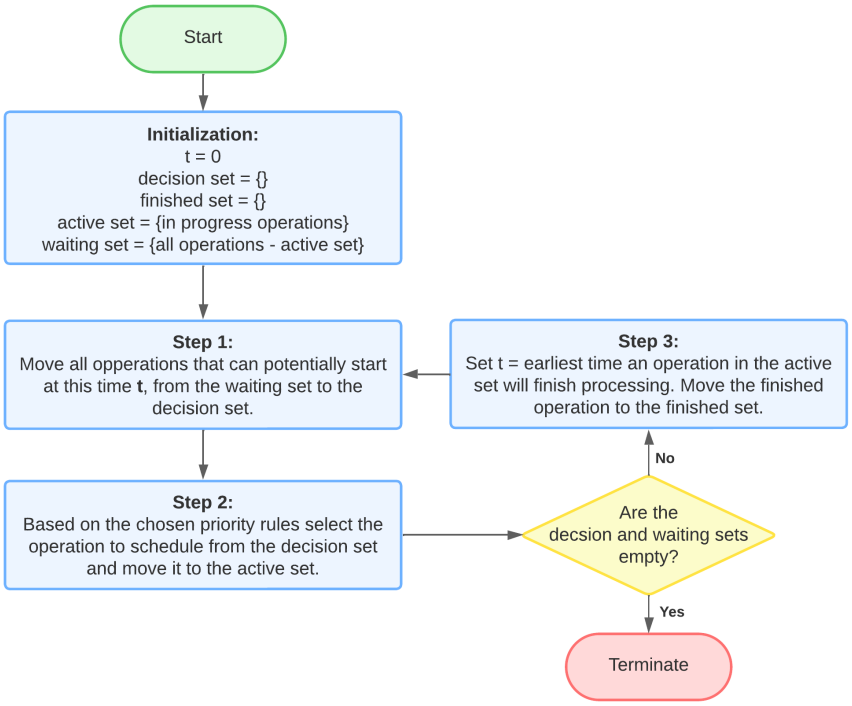



FIGURE 2.3: Flow chart of detailed scheduling process

Firstly, there is the possibility to cluster operations that are to be processed on the same machine/ workstation. If so specified by the planner the operations pertaining to a cluster are then delayed until all the operations belonging to that cluster arrive at the machine. An example of why one might want to consider *clustering* is to minimize waste/ scrap or to reduce setup time, e.g., in metal shearing processes, parts of different shapes can be punched or cut from the metal sheet, reducing material waste.

*Load balancing* is also a feature that can be incorporated in the scheduler. This helps choose the specific machine or employee for an operation in cases where there are multiple possible machines and employees. The balancing can be done based purely on the load, or based on job characteristics like size. For example when there are two of the same machines but one is setup to handle smaller materials and the other to handle larger materials. In this case load is balanced based on size as you'd schedule the operation on the machine that best fits its material size.

Another special case is when an employee can operate multiple (non-)identical machines concurrently. Think for example of cases where an employee is only required to setup the machine for the operation while the machine can then finish the operation unsupervised. In this case the employee can setup machine 1, have it start the operation and then go to machine 2 to do the same. The number of machines an employee can operate at the same time depends on how much of the operation requires an employees attention. This is encapsulated by the *manned percentage* parameter of the specific machine. Which expresses a percentage of time that an employees attention is required. An employee should for example be able to handle two operations simultaneously that require 30 and 70 manned

percentage respectively.

Oppositely, sometimes an operation or machine requires multiple employees at the same time, in which case the order needs to be delayed until the required number of operators and tools are available. Also, as in practice employees are of different skill levels for specific operations, it makes sense that some employees can perform tasks faster or better than others. This is something the the scheduler can take into account by linking it with the *employee skill matrix*.

Another thing that can be considered is *batch processing*, in which an operation is divided smaller batches over multiple machines and employees that can work simultaneously. Furthermore, there can be cases in which an employee is specifically assigned to an order, and the employee has to follow and care for that order from start to finish, throughout all its operations. Finally, preventive maintenance activities can be taken into account in cases where this applies.

## 2.3 Machine capacity

Another function of Limis Planner are the calculating and graphing of machine capacity. For each machine, both the daily (or weekly depending on the setting) maximum available capacity as well as planned capacity are calculated and graphed. This allows the user to see the planned and available capacity on each machine, well ahead of time. This gives the user a sense of understanding of the machine utilization's as well as capacity left available for new potential orders. Figure 2.4 shows and example of a machine capacity graph in Limis Planner.

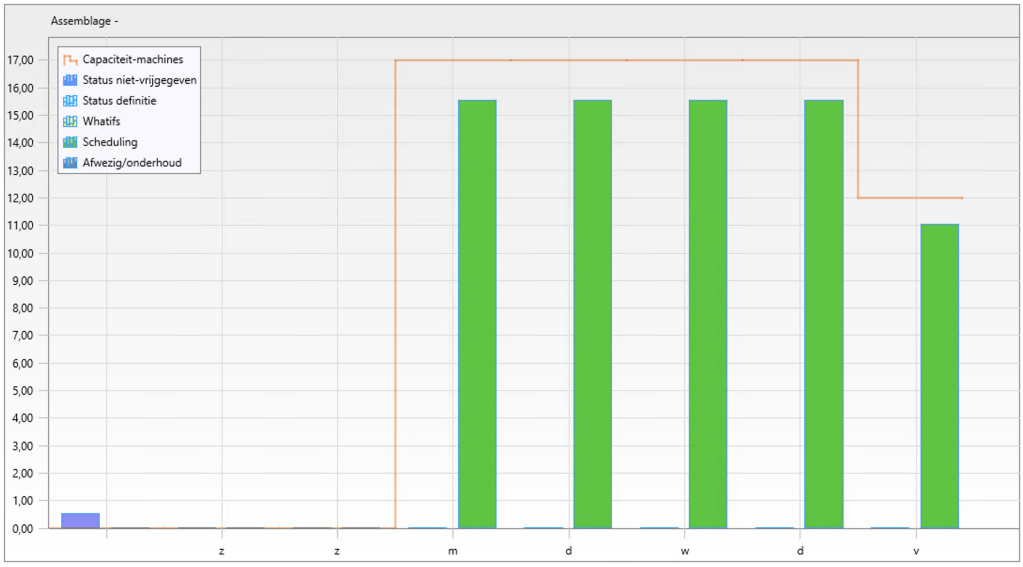



FIGURE 2.4: Example of a machine capacity graph for one week in days

Aside of the planned capacity there might also be overdue work that still had to be completed. The first bar from the left shows the amount of hours of overdue work for the specific machine. The orange line indicated the maximum available capacity per day and respectively the bars the planned capacity that day.

Some machines may inherently not be able to operate continuously but otherwise the limiting factor in the maximal daily machine capacity are often the employee working hours. While some machines may fully or partially operate autonomously, many require an employee to perform a task. Partially may be for example just an employee doing setup and the machine performing the rest of the task unsupervised.

## 2.4 Order acceptance in Limis Planner

### 2.4.1 The current situation

Order acceptance has its own tool within Limis planner called Limis orderacceptatie. This tool in its current use server mostly as a "sanity check", in that it serves as a layer between orders being placed and orders being included in scheduling. This in the sense that it will often quickly become apparent if there are either errors in the input or if the order is just otherwise incorrect, so that they are handled before being incorporated in the scheduling.

Notably, there seems to be a disconnect between order acceptance in literature and order acceptance as used in Limis planner. Whereas order acceptance in literature is mostly concerned with actually choosing whether or not to accept a job, often based on the idea of leaving capacity available for potentially incoming more profitable orders. Order acceptance at Limis customers the order acceptation decision is in practice almost always a yes, and its much more about either determining a feasible due date or determining impact on existing orders when adhering to the requested due date. It is in that sense, from a theoretical viewpoint, in essence more of a capacity planning, rescheduling or due date assignment problem.

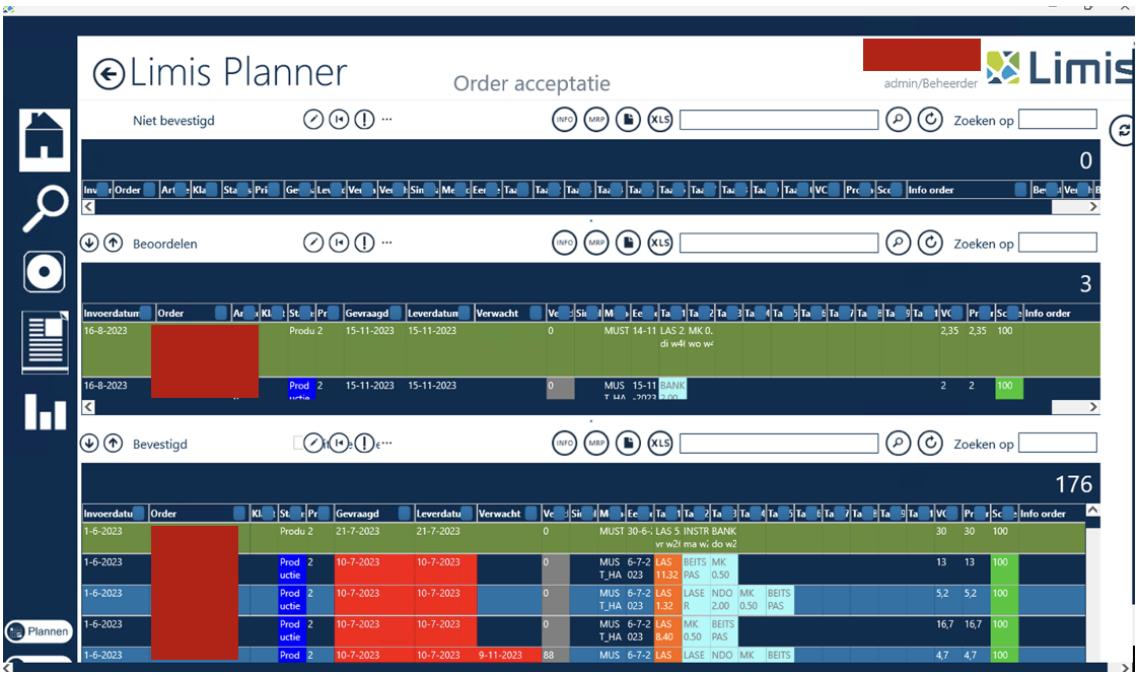



FIGURE 2.5: Screenshot of the current order acceptance tool within Limis Planner

In its current form all the calculations that are part of this order acceptance tool are

based on the full scheduler, i.e., the entire scheduling problem is run again including the new order. Essentially order acceptance currently comes down to performing another run of the scheduler and looking at the resulting due date for the order under evaluation. The schedule created through the order acceptance tool is separate to the original schedule created by the scheduler so it is not actually affecting the current schedule in use.

Despite being inefficient this is still feasible for smaller clients as their scheduling problems are not of a problematic order and do not take long to solve. For anything larger than the smaller clients however this quickly becomes unfeasible as this can take multiple hours for a single run, during which new orders may have already arrived again. Therefore there is need for a new approach, such as proposed in this report.

### 2.4.2 The desired functionality

As mentioned there clearly was a disconnect between what OA is in literature and what they were currently doing. Over time, through various discussions with Limis' executives and employees, fueled by knowledge obtained trough the literature research, we came to understand what we actually wanted to achieve.

The basis of this is that we want to quickly find out if we have the available resource capacity available to fit an order before a requested due date. Because with the way the scheduler works in Limis Planner we don't need to know exactly when which operation is scheduled when and how, we just need to know if we have the materials and man/ machine capacity available for each operation in an order given their sequence of processing. Because if we do, then the scheduler should have no issue scheduling this order before the requested due date, without (significantly) hindering existing commitments/ orders. In case the order does not fit we want to either find out what the issue(s)/ bottleneck(s) are, and what capacity adjustments need to be made to overcome these. Or if the order is less important, we want to estimate the earliest alternative due date at which we can deliver the order, without having to make any capacity adjustments. Another use case for the latter part is when one does not have a desired due date but simply wants to know what the earliest possible due date is, without affecting the current schedule.

So essentially we're after a quick assessment of how the order fits on top of our already scheduled work load. How and where exactly this will end up in the actual schedule is of no real concern at this stage, and will be handled by the regular scheduler in Limis Planner later on.

## 2.5 Conclusion

Limis Planner users can utilize the scheduling function for planning orders, employees, machines, and tools. They receive customer orders that arrive dynamically and that follow either a string-type or assembly-type routing. These customers have workstations consisting of one or multiple (identical) resources, each with possibly different daily (available) capacities and processing times. The scheduling process involves two stages: rough planning, which employs backward infinite loading, and detailed scheduling, which relies on priority dispatching rules. We've seen how the scheduler works and the kind of other data that is available to use within Limis Planner, such as planned and available machine capacity. Furthermore we've described the current order acceptance tool, how it works, and

what its shortcomings are for its intended purpose, which is more complex than you might initially expect. The main issue with the current order acceptance tool is that it requires the full scheduling procedure to be run, which can take a lot of time to complete, especially for larger customers. In conclusion, we now know the state of the current situation, what we have to work with going forward and what we want to achieve with our order acceptance tool.

# Chapter 3

# Literature Review

In this chapter we review the literature related to the order acceptance problem at hand and its possible ways of dealing with it. Section 3.1 discusses relevant order acceptance problems. Section 3.2, 3.3, and 3.4 respectively discuss due data assignment, rescheduling and capacity planning approaches related to our problem. Section 3.5 reviews the literature on the possible role AI based methods can play in these approaches. Finally in section 3.6 we summarize and draw conclusions based upon the reviewed literature, by the end of which we'll have a clear view of the possible approaches to tackle our problem as proposed in literature.

## 3.1 Order acceptance

Order acceptance can be described as the entire process surrounding decisions regarding the acceptance or rejection of incoming customer orders, as well as determining the due date, quantity, and price. These decisions are necessary when they are not explicitly provided in the customer's request or are open to negotiation. The order acceptance decisions can be made periodically for a batch of incoming customer orders or in real time for each individual order as it arrives.

The decision to accept or reject orders is influenced by several factors, including the strategic direction of the firm, the existing capacity allocation, and the profitability of the specific order. In particular, there exists a trade-off between the revenue generated by an order and the costs associated with its processing. These costs may include potential delays for other orders and any penalties imposed if the order is delivered beyond its agreed-upon due date.

In line with the characteristics of Limis' customers, we conduct this literature review with a focus on order acceptance in make-to-order (MTO) or hybrid make-to-stock/make-to-order (MTS/MTO) job shops. These job shops experience dynamic arrival of customer orders, and there is a lack of information about the orders before they are arrive. The subsequent sections of this study will primarily examine orders with a string-type routing, consisting of operations that require uninterrupted processing on specific resources for designated durations, unless specifically mentioned otherwise.

After conversations with Limis executives it quickly became clear that we are, unlike most order acceptance literature, not really concerned with the acceptance/ rejection aspect, as Limis Planner users in practice typically accept all orders that do not already intuitively seem infeasible. Rather we are concerned with determining the capacity or schedule

adjustments that need to be made in order to make an order fit before a proposed due date or determining an alternative due date. Subsequently we identified the topics of due date assignment, rescheduling and capacity control as topics of interest relevant to our problem. Theses topics and the literature research towards them are discussed in the next subsections of this chapter.

## 3.2 Due date assignment

In due date assignment (DDA) models, manufacturers determine and quote due dates for incoming orders instead of customers requesting due dates. Either because the customer didn't request a due date or the requested due date is infeasible for the manufacturer. Quoting a due date for an order requires balancing between potentially order winning by being able to deliver quickly and on the other hand customer satisfaction by being able to adhere to the quoted due date. If you were to quote a early due date but you end up deliver (significantly) later than the promised due date, the customer will not happy.

Due date assignment problems can be seen as the the problem of predicting order flow time, i.e., predicting how long it takes you to process the entire order and possibly ship it to the customer. According to Kingsman et al. (1989) the release time $r_i$ of an incoming order i is determined by the arrival time of the order, the completion of design work if required, and the material availability. DDA models can be divided over two distinct categories; Rule based methods and finite capacity loading methods. In literature on DDA models, orders typically arrive dynamically and one at a time and performance is assessed by means of simulation study.

### 3.2.1 Rule-based due date assignment

Keskinocak and Tayur (2004) conducted a comprehensive review of the literature on rule-based Due Date Assignment (DDA). Their review primarily revolves around proposing new DDA rules and examining the performance of these rules under different operational scenarios, including priority dispatching rules and utilization levels. The performance evaluation measures employed in these studies include average and standard deviation of lateness, tardiness, and flow times, as well as the percentage of tardy orders. Based on their analysis, Keskinocak and Tayur (2004) identified a list of commonly used rules and derived some key insights and conclusions from the reviewed studies.

The following are some of the most common used flow time $f_i$ estimation rules, for an incoming order $i$, based the status of the (job shop) environment at the time of arrival:

Slack (SLK): $f_i = p_i + k_1$

Total work (TWK): $f_i = k_1 \times p_i$

Number of operations (NOP): $f_i = k_1 \times n_i$

Jobs in system (JIS): $f_i = k_1 \times p_i + k_2 \times JIS$

Workload in system (WIS): $f_i = k_1 \times p_i + k_2 \times WIS$

jobs in queue: $f_i = k_1 \times p_i + k_2 \times JIQ$

workload in queue: $f_i = k_1 \times p_i + k_2 \times WIQ$

TWK + NOP: $f_i = k_1 \times p_i + k_2 \times n_i$

Where:

$f_i$ = Flow time estimate of order i.

$p_i$ = total processing time of order i.

$n_i$ = number of operations in order i.

$k_1, k_2$ = empirical parameters.

$JIS$ = number of orders at the shop.

$WIS$ = total workload of orders at the shop.

$JIQ$ = number of orders in queue for workstations required by order i.

$WIQ$ = total workload of orders in queue for workstations required by order i.

There are seemingly some common conclusions found in literature on rule based DDA models as supported by Ragatz & Mabert (1984). First of all, rules that consider both order characteristics and the status of the shop/ environment (e.g. JIS, WIQ), perform better than those that just consider order characteristics. (e.g. TWK, NOP). Secondly, the performance of DDA rules also depends on the used dispatching rule(s). Finally, the performance of DDA rules is also dependent on utilization levels throughout the shop. More specifically, unbalances in utilization levels, lead to increase variability which in turn lead to decrease in performance of DDA rules.

### 3.2.2 Capacity loading based due date assignment

In finite loading, the rolling time horizon is typically divided into multiple discrete equal-length time periods. The operations of an incoming order i are loaded on top of the existing workload in the load profiles of the corresponding resources taking into account capacity availability and the precedence constraints. This means existing orders are not reloaded when a new order arrives.

$(C)WL_kt$ is the (cumulative) workload of resource k in period t, while $(C)C_kt$ represents the (cumulative) capacity of resource k in period t. Some studies on finite loading work with a capacity loading limit (CLL). This means that $WL_kt$ is allowed to be less than or equal to a percentage of the regular capacity $C_kt$, where CLL can be higher or lower than 100%. Each operation j of order i is assigned a due time $d_ij$ and the due date of order i is set to the due time of its last operation.

Some studies also use a minimum waiting time MWT for each operation as explained below. CLL and MWT are empirically determined parameters meant to improve the accuracy of assigned due dates. In order to ensure that capacity is consumed as planned by the finite loading method, detailed scheduling in the studies is based on the ODD priority dispatching rule, as explained in Section 2.3.2, which means detailed schedules are non-delay schedules.

Thurer et al. (2013) present a forward finite loading (FFL) method in which each operation is loaded fully within a single time period. To load operation j of an incoming order i on the required resource l, it first finds the period h in which the due time of the predecessor plus the processing time of the operation ($d_{i,j-1} + p_{ij}$) falls. If there is enough capacity available to load the full workload of the operation in period t, so $WL_{lh} + p_{ij} \leq CLL \times C_{lh}$, the operation is loaded in that time period. If this is not the case, the next period (t+1)

is considered until a period in which the operation can be fully loaded is found. the due date of operation j belonging to order i ($d_{ij}$) is set to the end of the time period in which the operation is loaded. Note that the release time of the order $d_{i0}$ is equal to 0. Figure 3.1 shows how an operation with a processing time of 5 is loaded according to FFL, with CLL = 100%. Although in this example h ($d_{i,j-1} + p_{ij}$) falls in period 3, the operation does not fit in its entirety in periods 3 and 4, and therefore it is loaded in period 5, which is the earliest period in which it does fit.

A key assumption in FFL is that operations are not allowed to be partially loaded,

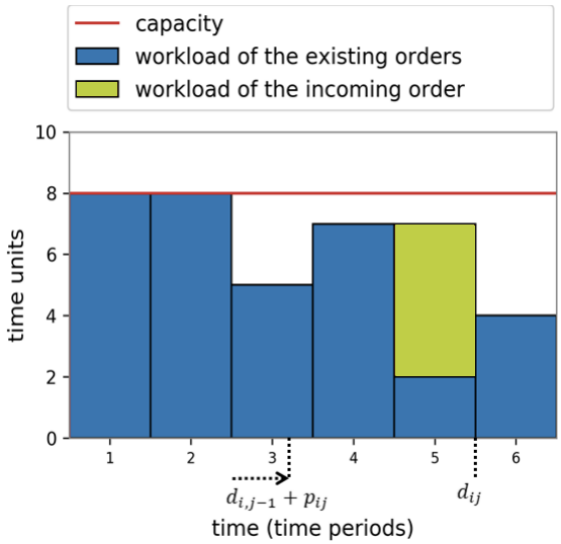


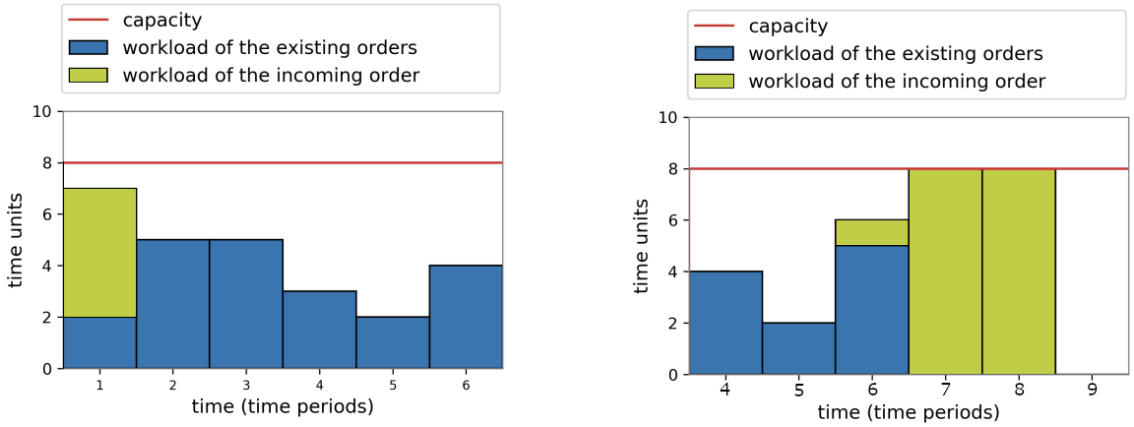

FIGURE 3.1: Forward finite loading example, adapted from Thürer et al. (2013)

i.e., spread over multiple periods (days). Therefore the maximum planned processing time per day can be no more than the length of the time period. Furthermore, the due time of an operation is set at the end of the day, as it is not known where within that day/ time period, the specific operation will be performed. A downside to this is that operations with a very short processing time will also be planned as if they take at least a full time period, which may cause over estimation of due dates.

Thurer et al. (2013) also discuss how to deal with deviations to planned workload of existing orders at the time of an order arrival. They differentiate between positive and negative backlog, where positive backlog is distributed over the time periods, and negative backlog is removed from the corresponding time period.

Bertrand (1983) proposes cumulative finite forward loading (CFFL), which instead considers cumulative workload and cumulative resource capacity. In order to load operation j, belonging to order i, on resource l, one first finds the period h in which preliminary due time $d'_{i,j}$ falls, where $d'_{i,j} = d_{i,j-1} + p_{ij} + MWT$. Starting from h, find the first period $q \leq h$, where the entirety of operation j can be loaded (in parts) over that period and subsequent consecutive periods. Then load the entire operation in period group m, where $m = q, q+1, ..., T$. If q equals h then the operation can be expected to finish at the preliminary due time. Otherwise the operation is expected to finish somewhere in m, so arbitrarily q + 0.25 times the period length is taken as due date.

Robinson & Moses (2006) propose a finite forward loading method in which operations are allowed to be partially loaded (FFLPL), i.e., operations do not have to fit in a single time period. They study and analyze the effect of the period length on the computation time and the accuracy of assigned due dates. They adapt the use of a granularity G as the period length. First the number of required period $b_ij$ is found using $b_{ij} = P_{ij}/G$. If $b_ij = 1$, then it simply finds the first single period ($q \leq h$) in which the operation can be fully loaded. If $b_ij > 1$, then it finds the first period h in which there is partial capacity available and the subsequent consecutive $b_{ij} - 1$ periods are full available. Figure 3.2 shows how the first two operations of an order are loaded using FFLPL, where $p_{i1} = 5$ and $p_{i2} = 17$. As can be seen in figure 3.2a the first operation can fully be loaded in period 1, therefore operation could potentially start in period 2. However, because periods 7 and 8 are the first fully available periods, operation 2 can not be loaded until period 6 as shown in figure 3.2b.




(A) (B)

FIGURE 3.2: Example of FFLPL for two subsequent operations, adapted from Robinson & Moses (2006).

## 3.3 Rescheduling

Rescheduling is the process of updating an existing production schedule in response to disruptions or other changes. Rescheduling studies have considered many types of manufacturing systems, including single machine systems, parallel machine systems, flow shops, job shops, and flexible manufacturing cells and systems. These disruptions can change the system status and cause significant deterioration to its performance, often requiring rescheduling of the the original production schedule to reduce impact. The following are the most common disruption factors identified in rescheduling studies:

- Machine Failure
- Job insertion / Arrival of urgent (rush) jobs.
- Order due date change
- Order cancellation

- employee absenteeism or shortage
- Delay in arrival or shortage of materials
- Change in job priority
- Rework or quality problems
- over- or underestimation of processing times

In our case we are only really interested in the job insertion disruption as this mimics that which we are concerned with; the impact of new jobs entering the schedule through the order acceptance tool. This may inherently include some change in job priority as the new order may be of higher priority than some existing orders, reducing their relative priority.

Vieira et al. (2003) present a framework for understanding rescheduling research. The framework includes rescheduling environments, rescheduling strategies, rescheduling policies, and rescheduling methods. Figure 3.3 shows this framework in which we have highlighted the parts that are relevant to this study.

In our case we are dealing with a finite set of orders to schedule, in which we do not consider stochasticity. Furthermore, we already have an existing schedule that we want to update under an event-driven policy, i.e., the arrival of a new order. This means we're also not concerned with schedule generation but only with schedule repair. Note that the schedules generated by Limis Planner are however, readily of robust nature, in that they include for example the extra wait time before and after operations to counteract possible minor disruptions.

| Rescheduling environments | | | | |
|---|---|---|---|---|
| Static (finite set of jobs) | | Dynamic (infinite set of jobs) | | |
| Deterministic (all information given) | Stochastic (some information uncertain) | No arrival variability (cyclic production) | Arrival variability (flow shop) | Process flow variability (job shop) |

| Rescheduling strategies | | | | |
|---|---|---|---|---|
| Dynamic (no schedule) | | Predictive-reactive (generate and update) | | |
| Dispatching rules | Control-theoretic | Rescheduling policies | | |
| | | Periodic | Event-driven | Hybrid |

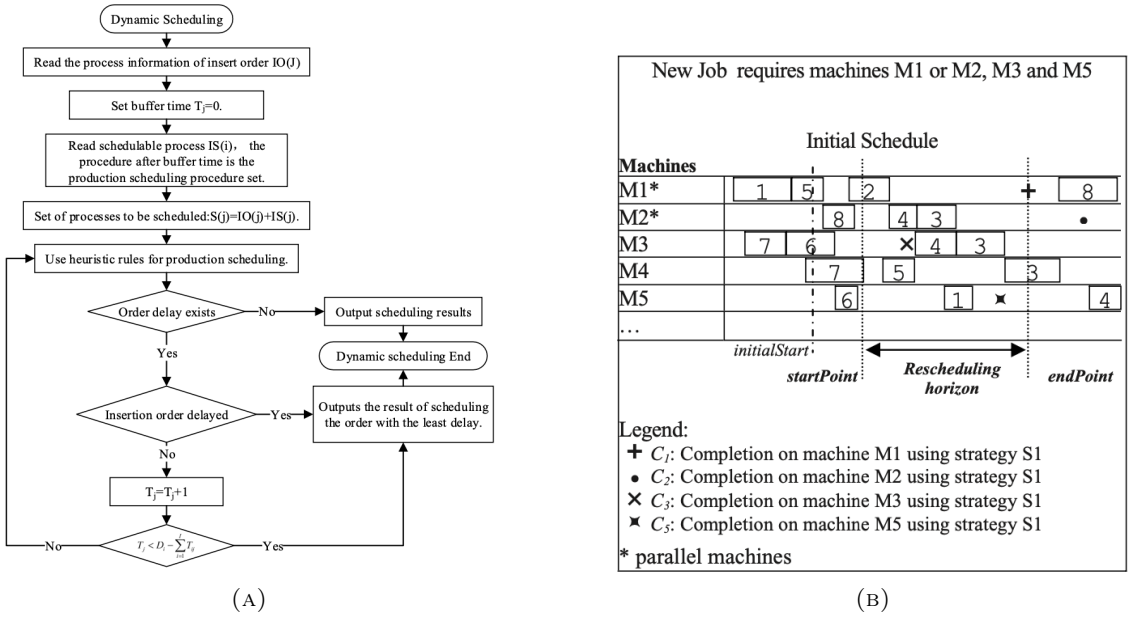
| Rescheduling methods | | | | |
|---|---|---|---|---|
| Schedule generation | | Schedule repair | | |
| Nominal schedules | Robust schedules | Right-shift rescheduling | Partial rescheduling | Complete regeneration |

FIGURE 3.3: Rescheduling framework

Huang et al. (2022) establish a progressive order insertion dynamic scheduling method for flexible job shops to minimize order delay. They propose a progressive rearrangement strategy which introduces a rearrangement buffer time to solve the problem of delays in production orders caused by insertion orders. They use a combination of SPT and EDD dispatching rules to reschedule everything after the buffer time. Figure 3.4a shows the

flowchart of the proposed progressive insertion strategy. If there are no delayed orders the scheduling results are printed, if there are delays, the buffer time is increased and scheduling process starts again. This is repeated until the inserted job is the one that is delayed or until the buffer time reaches the critical start date or the inserted job. This then indicates that the production capacity cannot meet the current order quantity and either the delivery dates or resource capacity must be adjusted. The approach is tested against total rescheduling. which shows that the progressive order strategy can effectively reduce order delay.

Moratori et al. (2008) investigate the problem of integrating new rush orders into the current schedule of a real world job shop floor. They introduce a number of match-up strategies that modify only part of the schedule in order to accommodate new arriving orders. The goal of the developed match-up algorithm is to define a time horizon within the original schedule that is to be rescheduled to accommodate the new order. The existing jobs in the time horizon as well as the new order then constitute a new smaller scheduling problem. Figure 3.4b shows an example of the match-up approach. After the newly defined problem is solved, the partial schedule contained withing the match-up horizon is replaced with the new schedule. It may be the case that the new schedule extends outside of the horizon, resulting in overlap. In which case the operations outside of the horizon are right-shifted. These match-up strategies are compared to right-shift and total-rescheduling. Analysis shows that they are comparable to right-shift rescheduling for stability, and as good as total-rescheduling for performance.




(A) (B)

FIGURE 3.4: Flow chart of the progressive insertion strategy (A), and example of match-up approach (B).

## 3.4 Capacity control

Thurer et al. (2013) propose the use of forward and backward finite loading methods. This includes (C)FFL as described in Section 3.3, and another rule-based method from the literature to decide if and on which resources capacity needs to be increased upon the

arrival of a new order. All incoming orders with customer-requested due dates are assumed to be accepted.

The procedure using FFL works as follows. An incoming order is loaded as described in Section 3.2. If the customer-requested due date can be met, the procedure stops and no capacity adjustment is made. Otherwise, capacity adjustments are made for the resource where the order had the longest throughput time, assuming that that the resource is most likely to be the bottleneck. Capacity increase starts in the first period. If an adjustment has already been made to that time period, the capacity in the next period is increased instead. If there is no time period without a previous adjustment before the due date of the operation, the procedure stops and the order is accepted without meeting the customer-requested due date. If an adjustment can be made in a time period, the order is reloaded and the above steps are repeated. Capacity adjustment in a period works by decreasing the processing times of the operations loaded in that period by a predetermined amount. I.e. if we reduce the operation processing time by 1 hour, and it then suddenly fits, we know that we need 1 extra hour capacity on that resource in that time period. Results of the conducted simulation study show that CFFL outperforms the other finite loading methods in terms of the average tardiness and the number of late orders.

Backward finite loading uses a procedure similar to FFL but in reverse, and except for determening operation due dates, it determines operation start dates $r_{ij}$. This means that the procedure starts at the final operation in the routing of the order and correspondingly the order due date. From there it goes back in time, trying to load each operation and thereby finding each operation a start date. The start date of an operation then becomes the due date for its predecessor in the order routing. If a start date, later than the current date, can't be found for the first operation of the order, the order does not fit between now and its proposed due date, without making adjustments to capacity. Figure 3.5 gives an overview of the studied capacity loading/ capacity control methods.

| Category | Acronym | Rule name | Brief description | advantages | disadvantages |
|---|---|---|---|---|---|
| Forward Finite Loading | **FFL** | Forward Finite Loading | Operation due dates are determined step by step, fitting operations to the remaining capacity. The planning horizon is broken down into time buckets. An operation is scheduled into the first time bucket with sufficient capacity after the previous operation due date and considering a minimum flow time allowance. The operation due date of the last operation determines the order due date. | Less computationally complex, highly responsive to short term changes, good for optimal resource allocation, does not require a known due date | risk of underestimating the overall work load, Potentially more inventory costs as products may be finished longer before their due date |
| | **CFLL** | Cumulative Forward Fnite Loading | As for FFL but operation due dates are determined by fitting the cumulative workload to the cumulative capacity. | Allows for more balanced workload over time. | less responsive to short term changes and computationaly more complex |
| | **FFLPL** | Forward Finite Loading with Partial Loading | As for FFL but allowing for partially loading operations. Operations do not have to fit in a single time period. | improved resource utalization, easier to fit operations | requires operations that can be interupted during processing |
| Backward Finite Loading | **BFL** | Backward Finite Loading | As for FFL but backwards. Operation start dates are determined step by step, fitting operations to the remaining capacity. The planning horizon is broken down into time buckets. An operation is scheduled into the first available time bucket with sufficient capacity. The start date of the first operation determines the planned release date. | Helps minimizing throughput time and in turn minimizing inventory cost by planning operations closers to their due date | requires a known due date, risk of suboptimal resource allocation, less capable of handling emergency orders or downtime as their is limited room before due dates |

FIGURE 3.5: Capacity loading framework as studied in this research.

## 3.5 AI based methods

This section describes some of the AI based methods encountered in literature related to possible approaches to our problem. This because the objective of this study was in part to explore the possible uses of AI with relevant application to Limis Planner. By far the most of the related literature that include AI based methods are about Scheduling

### 3.5.1 (Dynamic) Scheduling

Aydin & Öztemel (2000) develop an intelligent agent based dynamic scheduling system, in which the agent selects the most appropriate priority rule according to shop conditions in real time. The agent is trained by an improved reinforcement learning algorithm. Results show that the trained agent performs better than tradional alternatives and is able to select the most appropriate dispatching rule in real time.

Zhang et al. (2022) studies dynamic job shop scheduling based on deep reinforcement learning for multi-agent manufacturing systems. They present a multi-agent manufacturing system, which integrates a self-organization mechanism and self-learning strategy. A multi-layer perceptron is employed to establish a decision-making AI scheduler, which intelligently generates and optimal production strategy to perform task allocation. Proximal policy optimization (PPO) is used periodically to improve its decision-making performance. Experimental results show that the proposed methods is capable of obtaining scheduling solutions that can meet various performance metric as well as deal with resource or task disturbances efficiently and autonomously.

Wang (2018) studies an adaptive job shop scheduling strategy based on Q-learning. A weighted Q-learning algorithm based on clustering and dynamic search is proposed to determine the most suitable operations and optimize production. Convergence analysis and simulation experiments indicate that the proposed adaptive strategy is well adaptable and effective in different (complex) scheduling environments.

Zhao et al. (2021) first transform a complex dynamic scheduling problem into a Markov decision process (MDP) to then apply reinforcement learning. They use a deep Q network (DQN) to improve the model over time. Ten well-known heuristic dispatching rules are taken as the action set of the DQN and are selected using a custom designed action selection strategy based on the "softmax" function. Ten benchmark job-shop test instances are tested in a simulation environment. The results show that the proposed algorithm performs better than a single dispatching rule or traditional Q-learning algorithm.

Gui et al. (2023) reflect on the use a deep Q-network and also apply a deep reinforcement learning method, in an effort to have the model select the most appropriate dispatching rule at any given moment, with the goal of minimizing the mean tardiness. They again first transform their dynamic flexible job-shop scheduling problem into a Markov decision problem after which a RL problem is formulated. A reward function related to mean tardiness is designed so that that maximizing the rewards is equivalent to minimizing the mean tardiness. They train the network using the deep deterministic policy gradient (DDPG) algorithm to select the appropriate weights, thereby aggregating the dispatching rules into a better rule. Results from numerical experiments show that the proposed scheduling method results in significantly better scheduling results than a single dispatching rule or the DQN-based method in dynamically changeable manufacturing systems.

Wang et al. (2022) formulate a new dynamic multi-objective flexible job shop scheduling problem (DMFJSP) to simulate a realistic production environment. Six dynamic events can occur in the problem; job insertion, job cancellation, operation modification, machine addition, tool replacement, and machine breakdown. three objective as considered including longest job processing time (makespan), average machine utilization, and average job processing delay rate. They design a novel dynamic multi-objective scheduling algorithm based on deep reinforcement learning to solve the problem. The algorithm uses two deep Q-learning networks and a real time processing framework to generate complete scheduling schemes. In addition, an improved local search algorithm is adapted to further optimize scheduling results. The results show the superiority and stability of the proposed approach compared to well known scheduling rules and standard deep Q-learning algorithms.

### 3.5.2 Rescheduling

Palombarini & Martinez (2012) propose the automatic generation and updating of rescheduling knowledge, using simulated transitions of abstract schedule states. Deictic representations of schedules are used to define a repair policy which generates a goal- directed sequence of repair operators to face unplanned events and operational disturbances. Diectic representations meaning that the schedules are accompanied by the current state of the manufacturing system. They consider the arrival of new/rush orders, material delay/shortage or machine breakdown events, using the SmartGantt prototype for interactive rescheduling in real-time. SmartGantt demonstrates that due date compliance of orders-in-progress, can be dramatically improved by means of relational reinforcement learning and a deictic representation of rescheduling tasks.

Palombarini & Martinez (2019) present a novel approach to model a real-time rescheduling task as a closed loop control problem in which an artificial intelligent agent employs its control knowledge. This knowledge is generated off-line using a schedule simulator to learn schedule repair policies directly from high-dimensional inputs, and is stored in a deep Q-network. The network is trained using Q-learning based on color-rich Gantt chart images and prior knowledge. The resulting control policy can be used to obtain optimal sequences of repair operators for unseen schedule states to counteract the effects of disruptions at the shop floor.

### 3.5.3 Capacity control

Windmüller (2020) provides a model for planning the usage of overtime and outsourcing for job shops with various reinforcement learning methods. Multiple reinforcement learning methods from literature as well as naive heuristics are compared. The results show that one of the proximal policy optimization performed very well and resulted in a 10% less total cost than the others. Showing that reinforcement learning can be a viable alternative to established heuristics.

## 3.6 Discussion

The literature review has shown that the studies towards the various parts of the order acceptance problem as observed in this study, do typically not consider material availability and instead generally assume that materials are continuously available. Whereas this is an integral part of our problem, with the exception of the few cases in which material

requirement can be disregarded.

Having reviewed methods of due date assignment, rescheduling, and capacity control, we find that there are plenty of proposed solutions in literature that can potential be adapted to fit (parts) of our problem. Backward finite loading seems to be a promising way of making the initial determination if an order can be processed between now and the proposed due date. This because we have a desired due date from which we can work back in time, trying to fit the operations based on the available capacity on each day. Furthermore, it seem that both a finite loading method as well as partial rescheduling could be used to determine capacity adjustments required to make an order fit, if the initial decision was negative. For orders of less importance either forward finite loading or right-shift rescheduling seem appropriate methods to determine an alternative due date if again the initial check gave a negative indication. Here forward loading would be used as opposed to backward loading, as we do not have a set due date, but rather are trying to find one. So by going forward from the current moment in time, fitting operations to the earliest possible day, we find the earliest date at which the order can be delivered.

With regards to the AI based methods reviewed, because of the original desire to employ a form of AI in Limis Planner, a couple of things can be concluded. First of all, simply going by the number of studies found in literature on the different topics, it seems that AI/ reinforcement learning is more relevant in scheduling (schedule generation) rather than rescheduling, i.e., there seem to be far more studies into AI based scheduling than AI based rescheduling. Looking at the context of these studies one can draw the same conclusion, that is, they do not match the context of our problem at Limis. Context here referring to characteristics of the problems, such as the type of orders, factory setups, etc. Since scheduling is outside of the scope of this study, we went on to consider methods of dynamic/ on-line scheduling and rescheduling.

The focus of the studies on AI based rescheduling is on determining which type or rescheduling to apply based on the characteristics of the shop at the time and specifically the type or disruption. We are however not interested in dealing with different type of disruptions in this research as our only concern in this case is the effect of new order insertion on the existing schedule. As for the reviewed literature on dynamic/ on-line scheduling these concern high tech autonomously operating factories, in which there is no predetermined schedule, but rather the individual machines are intelligent agents that determine on the spot which order to process next. This nothing like the manufacturing environments encountered at Limis Planner users, i.e., Limis customers do not use such sophisticated advanced production systems. Therefore this is seemingly not relevant for Limis at the current moment in time.

An approach that was not found in literature, but seemingly could be used in the initial decision on whether or not an order can be processed before the proposed due date, is the use of classification models. So using a machine learning method rather than a reinforcement learning method, i.e., making a prediction based on characteristics of the shop, the incoming order, and the current schedule. This seems to be the only (minor) application of an AI based method that could be relevant to our problem. Therefore we will test this along the proposed backward finite loading method as will be further explained in chapter 4.

So, as we've determined scheduling to be outside of the scope of this study, we can conclude that reinforcement learning based methods are not as relevant to the problem studied in this thesis as initially thought and hoped. We therefore focus on the non-AI based methods, as further explained in the next chapter on solution design. The exception being the testing of a simple classification model technique to make an initial prediction on whether an order will fit before a proposed due date or not.

# Chapter 4

# Solution design

Having studied literature on possible approaches, we can now formulate the proposed solution to our order acceptance problem. This chapter will present, discuss and evaluate the proposed solution. We start in Section 4.1, by formulating and discussing the proposed solution. We begin with an overview of the entire approach and follow up with detailed explanations of each step of our approach applied at the different stages. In section 4.2 we go over the assumptions and simplifications. Section 4.3 explains our use of classification models as an alternative approach In section 4.4 we discuss the resulting tool that is meant to bring the proposed method to practice. Finally, section 4.5 discusses the integration of the tool into the existing software (Limis Planner).

## 4.1 Proposed solution

In this section we formulate and discuss the proposed solution. We start by going more in depth on the problem. We then give a broad overview of the proposed solution and follow this up by going more in depth on each of the separate stages of the solution.

### 4.1.1 The problem

The problem at hand can be separated into two phases and the second phase into two different sections. In the first phase we check whether the order fits before the requested due date, i.e., we want to find out if we have the capacity and materials for each of the orders task before the due date. If this is the case, the order acceptance decision is positive and the second phase can me omitted. If the result from the first stage is negative, i.e., the order does not fit before the requested due date, we get to the second phase.

To be clear, we say an order "fits" if, and only if, all of its operations can be scheduled between the due date and now. Using backward loading this means that the first operation of an order can be scheduled after $T_0$ (the current date). Figure 4.1 shows an example of this for two orders, one that fits and one that doesn't fit. The figure shows a Gantt chart for three machines (M1, M2, M3) and two orders with the same due date, both consisting of 3 operations. As seen in the figure, Order 2 fits, as its first operation can be scheduled to start at a time later than $T_0$ (today). Order 1 however, does not fit, as its first task would have had to start before today for the full order to be finished by its due date. A different order of scheduling the orders might have resulted in a different outcome, however that is of no concern to us. As things stand, in the example, order 1 does not fit, that does not mean it can't fit in a different schedule, but with the schedule

as it is, it simply does not.


FIGURE 4.1: Order fitting example

The second phase depends on the order in question, namely whether it is a so called "must-have" order or a "best-fit" order. This comes down to whether the order is important enough to justify taking action to make the order fit on the requested due date, or not. So if the order is a must-have order, we want to find out what the bottleneck is. i.e., what is preventing us from being able to fill the order on the requested due date, and how to overcome this. An example could be that 4 extra hours of machine x are needed on a certain day, or that materials somehow need to be procured before a certain date. If the order is a best-fit order we instead want to find the earliest date that we can finish the order by, without taking action like making capacity adjustments etc.

The proposed solution to this problem needs to work with the data available within Limis planner, as explained in chapter 2. It also needs to work independent from factory characteristics, i.e., it should work for all of Limis customers that provide sufficient data. Furthermore, it should be able to do this without having to make adjustments to/ reprogram the tool. This means that the tool resulting from the solution should be able to automatically read out the data from the customer's database and setup the environment accordingly. A solution should also not involve scheduling / making changes to the scheduler. This is a condition set by Limis executives at the start of this project. But above all, the solution needs to be able to produce a result in a short time span ($\leq$ 2 min.).

### 4.1.2 The solution

We use finite loading methods as a basis for our solution approach. For the initial check on whether the order fits we employ backward finite loading. Unlike backward finite loading approaches encountered in literature we do take into account material requirements by checking if the planned date is farther out than the material's lead time. If the result of the initial check is positive, meaning the orders fits, as defined before, the procedure can be stopped, which helps reduce the running time.

If the result is negative, meaning the order doesn't fit, and if we're dealing with a must-have order we employ a capacity control method in which we reduce the processing time of a chosen operation, e.g. with the longest throughput time by a set amount and run the check again. This is repeated until the order fits, each time time the operation is chosen according to a given selection rule/method. In chapter 6 we experiment with different methods of choosing the operation to adjust as well as the amount of time to decrease the operation by in each step. All the required processing time reductions are recorded, along with their respective machines and scheduled dates. These then tell us what capacity adjustments we have to make to which machine on which date(s).

If instead the order is a best-fit order, a finite forward loading method is applied to find the earliest possible due date for the order, without any capacity adjustments. This again takes in consideration material lead times, which speeds things up as it does not have to search in time periods before the current date plus the lead time.

The outcome of this proposed approach in its entirety is one of three things:

1. The initial check finds that their is no issue and the order can be accepted as is.
2. The capacity control methods finds a minimal capacity increase that is needed to make the order fit.
3. The forward loading method finds the earliest date at which the order can be delivered without capacity adjustment.

Figure 4.2 shows and overview of the entire proposed solution including the proposed approaches for each stage. The following subsections will discuss these approaches in dept per stage. We propose a new order acceptance tool, that performs all these approaches as required depending on the order. The actual tool itself and how to use it will be discussed in Section 4.4, whilst this section continues discussing the underlying procedures.

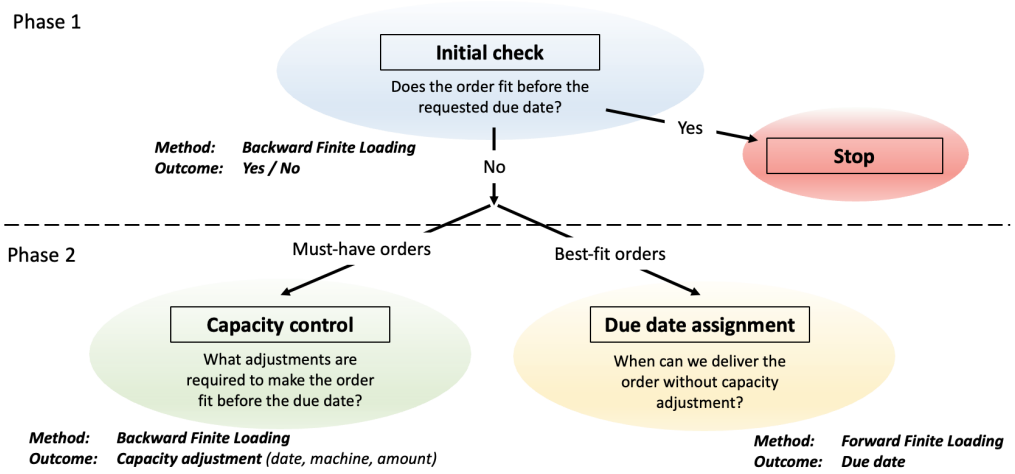



FIGURE 4.2: Overview of the proposed solution method(s)

Figure 4.3 shows the flow chart for the entire proposed procedure, from beginning to end. This includes phase 1's initial check, and both of phase 2's procedures, depending on the type of order. Note that the flow chart speaks of tasks, this can be read as operations (belonging to an order), as we use these terms interchangeably here. As mentioned before, the capacity control procedure selects an task/operation to adjust based on the selection rule, e.g., longest throughput time (LTPT) or highest order level (HOL). In Chapter 6 we conduct experiments to determine which selection rule one should use, as well as which step size should be used, i.e., by how much the chosen operation's processing time should be adjusted in each step.

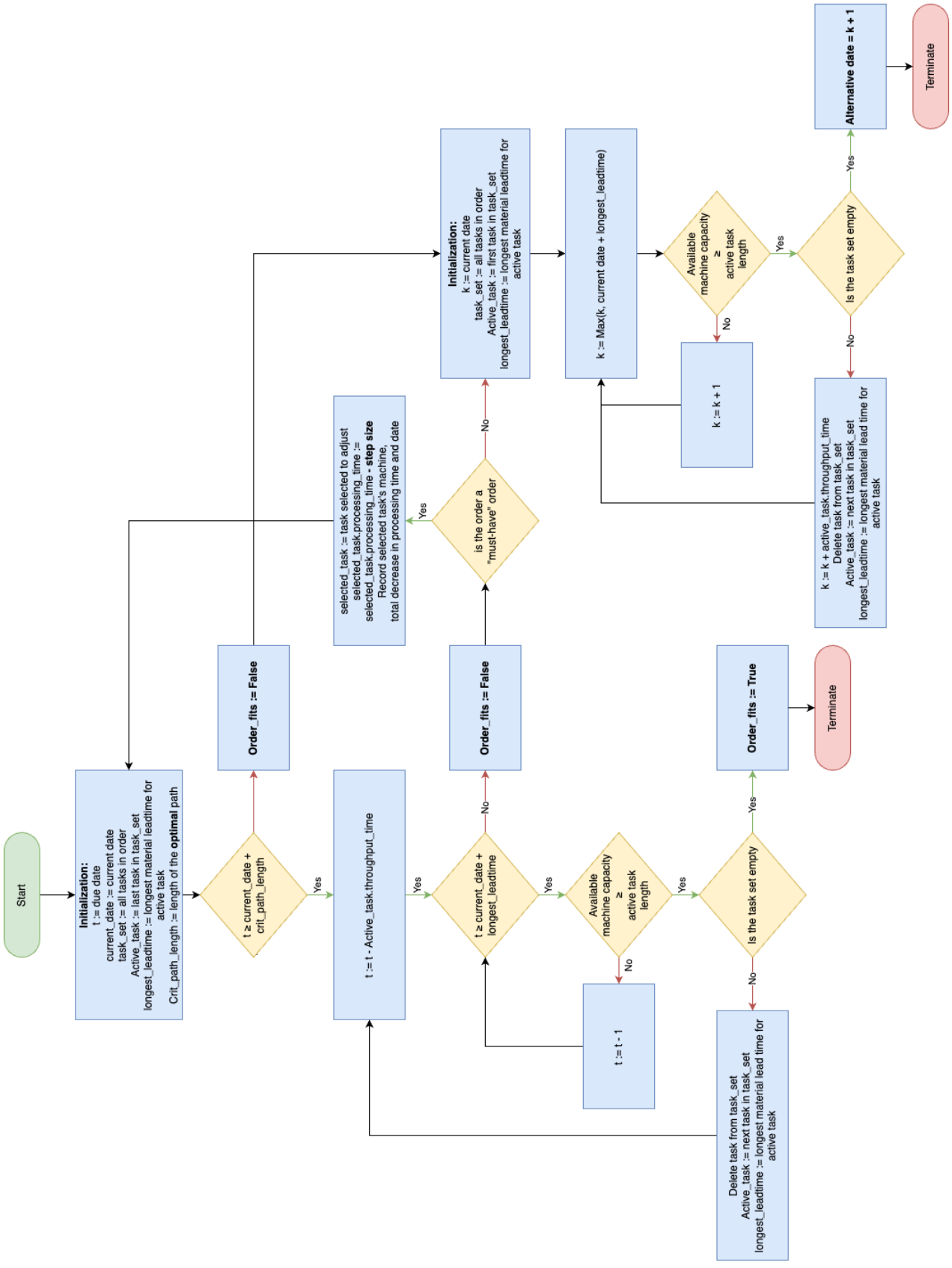



FIGURE 4.3: Flow chart for the entire proposed procedure

### 4.1.3 Phase 1: The initial check

The initial check will be done using backward finite loading (BFL) as previously described in the literature review section. The procedure starts at the desired due date with the last operation in the order and works its way back towards the current date. It tries to find the earliest day on which the machine required by the operation has enough free capacity to fit the operation on. The earliest here meaning the closest from the due date, so the furthest away from the current date. If it finds a suitable day for the operation it moves to the operation before that and starts searching for a suitable day for that operation starting at the day before the just scheduled operation. While doing this it constantly checks if the date it is considering, is further out than the lead time of the materials required for the operation. This continues until the first operation in the order is planned before the current date, in which case the order fits, or until the current date is reached, in which case the order doesn't fit. Depending on the chosen settings the check will either stop and return a negative result when a materials lead time is not met, or the procedure takes note of the conflict and continues its process. More info about this and other settings can be found in chapter 5.

Figure 4.4 shows the flow chart of the initial checking procedure as employed by the tool. The outcome of this initial checking stage is either a positive recommendation or a negative one. A positive result means that the order should fit before the deadline without issues, i.e., running the scheduler should result in the order being scheduled before the requested due date. In this case the procedure ends, whilst a negative result means the procedure continues with either finding the capacity increase required or an alternative due date.

We choose to use backward loading instead of forward here, because we aim to complete orders just in time, minimizing throughput time where possible. Furthermore, starting orders closer to their due date minimizes unnecessary time spent in storage, reducing storage costs. Finally, pushing jobs further back where possible, leaves more room for flexibility in the short term, as unforeseen circumstances may affect the day to day manufacturing schedule, i.e., if there is more spare room in the short term, it is easier to fit in emergency orders or adjust for unforeseen delays and problems.

We also test using a machine learning model to predict whether an order fits, as an alternative to the backward finite loading. This is further discussed in its own section, Section 4.3 and the related experiments and their results in chapter 6.

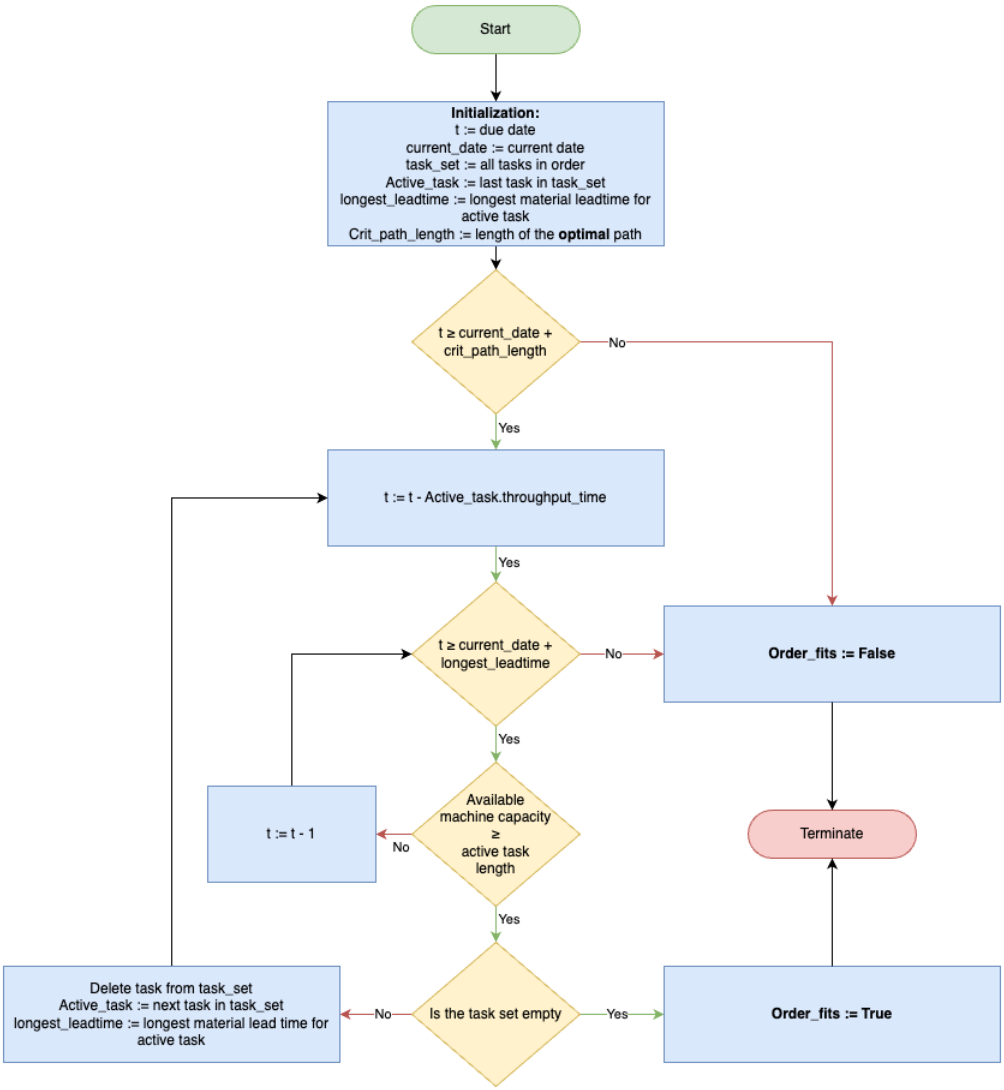



FIGURE 4.4: Flow chart for the initial checking algorithm

### 4.1.4 Phase 2: Must-have orders (Capacity control)

The capacity control/ fitting method employed to find the capacity adjustment(s) required to fit the order before the requested due date, is based on the same backward finite loading method used by the initial check. In fact it is essential continuously rerunning this check with slightly adjusted capacities each time. The method consists of the following steps:

1. Select an operation to adjust (e.g. longest throughput time)
2. Reduce the processing time of this operation by an amount (step size e.g. 5min.)
3. Run the initial check again (backward loading procedure)
4. Record the date on which the adjusted operation is now planned and its total decrease in processing time.
5. If the entire order now fits, end the procedure, else go back to step 1.

So the procedure first selects an operation to adjust, based on a selection rule, e.g., the LTPT rule finds the operation with the longest throughput time. Then, instead of increasing the machine capacity on a specific day we decrease the processing time of an operation (in the order) to simulate an increase in capacity without having to adjust each separate day. The principle here being that it does not matter if you subtract an amount of time on the left side of the equation, or add that same amount of time on the right side of the equation.

If an order with decreased processing time then suddenly fits on a day that it previously didn't, we know that we need an increase of capacity on that day, by the amount the processing time was decreased. This means that we do not have to perform a run for each individual day that we want to try an increased capacity on. This process is repeated until the entire order fits. Along the way the decreased processing times and the days on which the adjusted orders are planned are recorded. The result of the procedure is all the adjustments required to make the order fit. Note that an outcome could be that multiple machines need increased capacity. If the first found increase is not enough to make the entire order fit before the requested date, it goes on to the next operation and so on, until eventually the entire order fits or we find that it is impossible to make the order fit, due to for example capacity increase limits set by the user, e.g., max 2 hours of extra capacity per day. Figure 4.5 shows the flowchart of this procedure.

The time with which an operation is decreased as well as how to select which operation to decrease the processing time of, is experimented with, as will be further explained in section 6.2. Aside of just the operation with the longest processing time we for example also test using the operation with the highest level within the order (HOL).

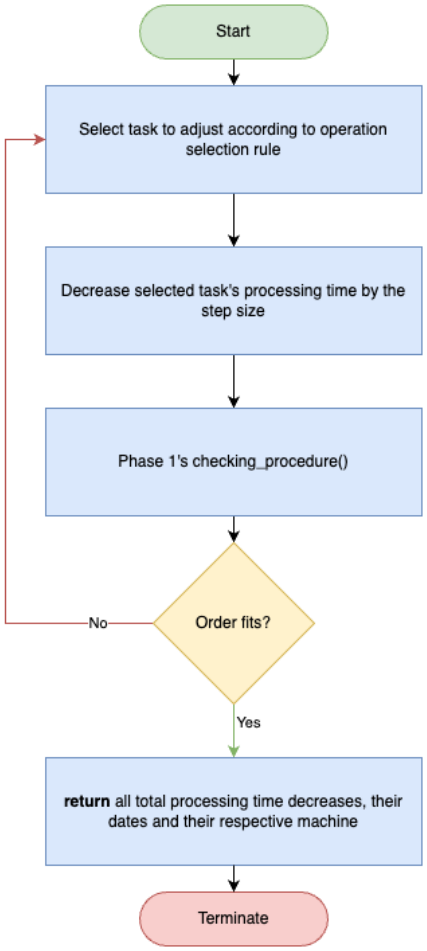



FIGURE 4.5: Flow chart for the capacity control (must-have orders) algorithm

### 4.1.5 Phase 2: Best-fit orders, (Due date assignment)

For the Best-fit orders we employ a standard finite forward loading method, with the addition of a material lead time check, if chosen in the settings. This means that the procedure starts with the first operation of the order, at the current date or the current date + the lead time for the materials required by the first operation, if no ample stock is kept. It then tries to find the first day on which the required machine has enough capacity available to fit the operation on. Once this day is found, it moves on to the next operation and repeats the process starting at the next day. This is repeated until the final operation of the order is planned. The resulting due date is the the day after the final operation is processed. This date is the first date on which the order can be delivered without making any capacity adjustments. Figure 4.6 shows the flow chart for this forward loading procedure.

Here we use forward loading rather than backward, for two simple reasons: First of all, we do not know the date we're gonna end up with, so we have no date to work backward from. Using backward loading here would mean that we would have to try many different dates through trial and error, meaning we would have to perform many runs. While with forward loading we only need a single run. This ties in perfectly with the second point, which is that we want to find the earliest possible date at which the order would fit, which is exactly what forward loading does, as it pulls every operation as much forward as possible.

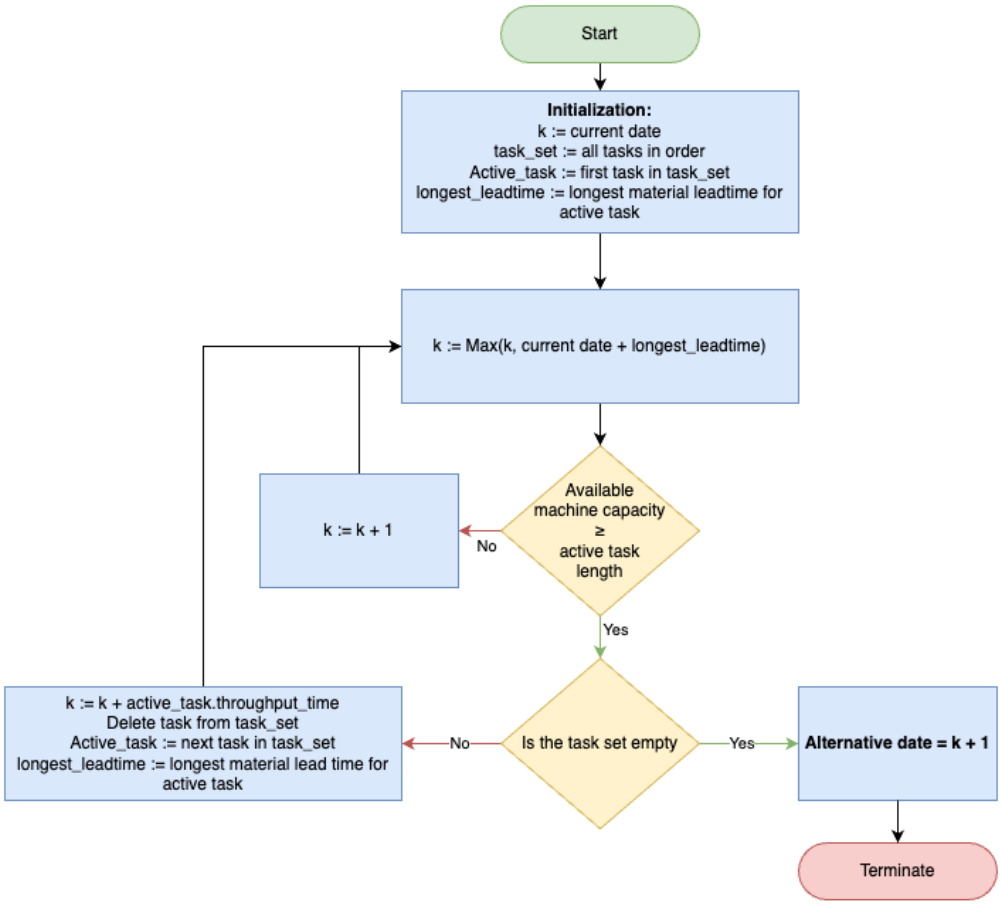



FIGURE 4.6: Flow chart for the forward loading (best-fit orders) algorithm

## 4.2 Assumptions and simplifications

We do not know when exactly on a day an operation is scheduled, therefore the succeeding operation can be scheduled the next day after at the earliest. The exception being operations that are specifically marked as "no-delay operations", which are operations that typically take mere minutes to perform. So essentially we count at least a day for each operation (or a multiple of days for operations taking longer than one day).

A notable simplification and limitation, is that we couldn't work with individual machine capacities, in case of multiple identical machines in a machine group. This due the fact that Limis does not store the capacities on an individual machine level outside of the scheduling process. Say for example that a machine group consists of 5 machines operating 8 hours on a day, we assume the capacity for that machine to be 40 hours on that day. So machine capacity is pooled between all identical machines unless a customer has specifically declared them as individual machine groups. Imagine for example that we have 5 hours left available on the machine group on some day. We do not actually know how the 5 hours is divided over each of the 5 machines. It could be that there is 5 hours left available on a single machine and none on the others, or there could be 1 hour available on each. Either way this does not make a difference in Limis Planner, whereas it could make a difference in practice.

Another assumption is that "external machines" have unlimited capacity, i.e., all operations that are handled externally can begin processing the the day after their predecessor has completed and simply always takes the given lead time to be finished and returned to the manufacturer. If for example an painting operation is required for a part that is handled by an external party, and the throughput time for this operation is given as 4 days, we assume that the part is back in our possession 4 days later, painted and all, ready to proceed with its next operation.

Another simplification is that we do not work with inventory positions in the order acceptance tool. Instead we just assume there to be no inventory, and work with the lead times of materials. For some operations it is specified that (certain) materials can be neglected due to the materials always having plenty of stock for example. Imagine something requiring a screw, of which thousands are kept in stock because they are constantly used. The alternative to this would be to simulate inventory levels over time, but this would both add a massive layer of complexity as well as drastically increase the computational effort of the procedure which would be counter productive to what we're trying to achieve. Also, despite being a simplification of reality, it is a step up from previous similar studies performed in which all materials are disregarded/ assumed to be continuously available.

## 4.3 Classification models

Because of Limis' explicit interest in using AI/ machine learning based methods, we explore the use of classification models as an alternative to the finite backward loading procedure in phase 1. Classification is a supervised machine learning method where the model tries to predict the correct label for given input data. Based on so called predictors it classifies an input as belonging to a distinct group (usualy one of two), as depicted in figure 4.7.

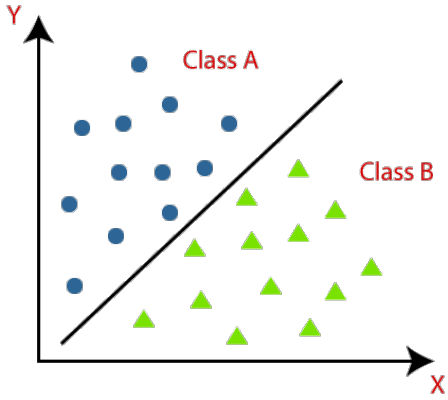



FIGURE 4.7: Example of classification into Class A and class B

This lines up nicely with what were trying to do in phase 1 of our approach, i.e., we want to determine whether an order fits or not. In other words, we want to classify a certain case (order with a given due date) as belonging to the class "fitting" or the class "not fitting".

We test 3 of the most common classification models; Logistics regression, decision tree, and support vector machines (SVM):

**Logistics regression** Logistic regression is a regression model used to determine the probability of occurrence of an outcome for an observation. In this case the probability of an order fitting is estimated based on the data given in the training set.

**Decision tree** A decision tree model is, as the name suggests, a tree-like model in which each node represents a test criterion of a feature. Each branch corresponds to a outcome of the test and each leaf node shows the class outcome. In our case this is for each order either 'Fitting' or 'Not fitting'.

**Support Vector Machine (SVM)** Support vector machines are supervised learning models with associated learning algorithms for data classification. These algorithms create a line or hyperplane that separate data into classes distinctly classifying all data points in an N-dimensional space. We test support vector machines using three of the most common kernels: linear, polynomial, and radial.

We test these models using the following features:

**Number of days until the due date** This is the number of days between now (the moment of evaluation) and the due date.

**Number of operations in the order** This is the total number of operations that need to be scheduled as part of the order.

**Length of the ideal critical path** This is the number of days that the order would at a minimum take in an ideal scenario, i.e., we can plan each operation right after another.

**Number of days until the due date - length of the ideal critical path** This is the difference between the number of days until the due date and the length of the ideal critical path.

**Average daily machine hours available before the due date** This is the average amount of hours available on each machine on each day between now and the due date.

**Average weekly machine hours available before the due date** This is the average amount of hours available on each machine for each week between now and the due date.

**Average daily percentual machine availability before the due date** This is the average percentage of total machine capacity still available on a daily basis, between now and the due date.

**Average weekly percentual machine availability before the due date** This is the average percentage of total machine capacity still available on a weekly basis, between now and the due date.

**longest material lead time** this is the longest of the material lead times.

**Number of days until the due date - longest material lead time** this is the difference between the number of days untill the due date and the longest material lead time.

**average operation throughput time** this is the average throughput time of the operations that are part of the order

**longest operation throughput time** this is the longest throughput time of all operations in the order.

**number of external operations** this is the number of external operations that are part of the order.

## 4.4 The order acceptance tool

In this section we present the proposed new order acceptance tool that brings the proposed methods to practice. It has served both as a tool for testing the methods as well as being a pretty advanced prototype for an actual tool for customer to use. In fact it is in its current state pretty much ready for a beta release, given that the basic functionalities are working, the interface is user friendly and it is more than capable of loading and running on live customer data from their data base. Figure 4.8 shows a screenshot of the proposed tool in its entirety.

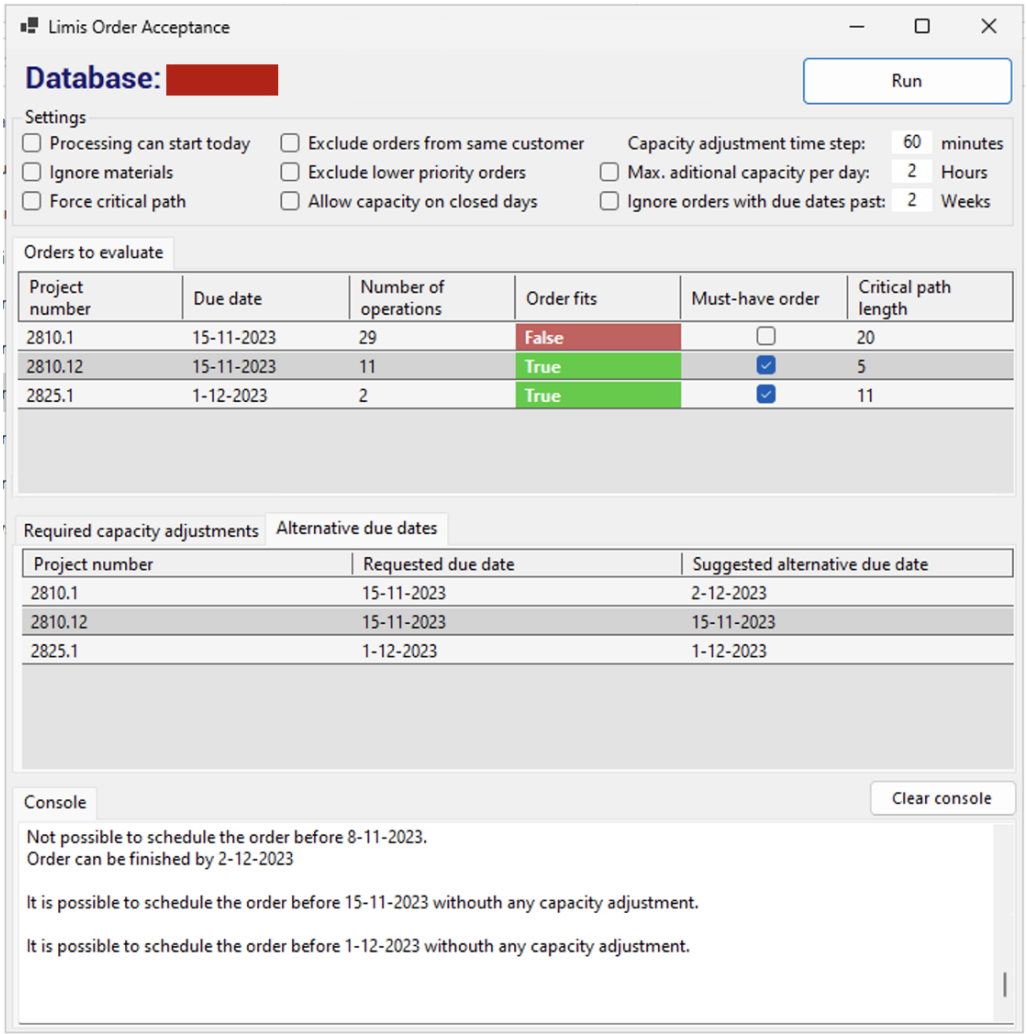



FIGURE 4.8: A screenshot of the proposed new order acceptance tool.

### 4.4.1 How to use the tool

The tool is extremely easy to use, and requires in basis only a few to a single button press to operate. When you start up the tool, all the required input data is loaded and setup correctly, e.g., machine data, order data, planned capacity, etc. At the top of the form you find the settings that you might want to use. The settings and when you might want to use them are further explained in the setting subsection.

Below the settings you see the "orders to evaluate" table, containing all the orders that need to be evaluated by the order acceptance tool. Each row of the table contains an order and an overview of its relevant information. This includes the project number, due date, total number of operations, whether the order fits, whether the order is a must-have order or not, and finally the critical path length. Clicking on the order number shows order structure/ routing, without including operations. Clicking on the total number of operations shows the same but including the operations and related info, like the required machine, planned date, etc. Further, you can edit the requested due date if you want to, and indicate whether the order is a must-have order or a best fit order by checking the Must-have checkbox. You can then run the order acceptance procedure for each order in the table by simply pressing the RUN button in the top right corner.

After hitting run, the entries in the order fits column, that previously read "Awaiting evaluation" will turn to TRUE or FALSE, and will be highlighted green or red respectively. This will immediately tell you if the order fits before the required due date or not. Then, based on the requested due date and the order type, one of two things happen. Either a required capacity adjustment, that allows the operation(s) to be expedited and thereby making the order fit, is found and is added to the capacity adjustments table as seen in figure 4.9a. Or an alternative due date is found and added to the alternative due date table as seen in figure 4.9b. The results are also written to the console at the bottom, for each order. As you can see, the required adjustments are shown as an amount of time in hours, on a machine, on a date, as part of a project (order). For the alternative due dates it just shows the new date, next to the requested date for each order.

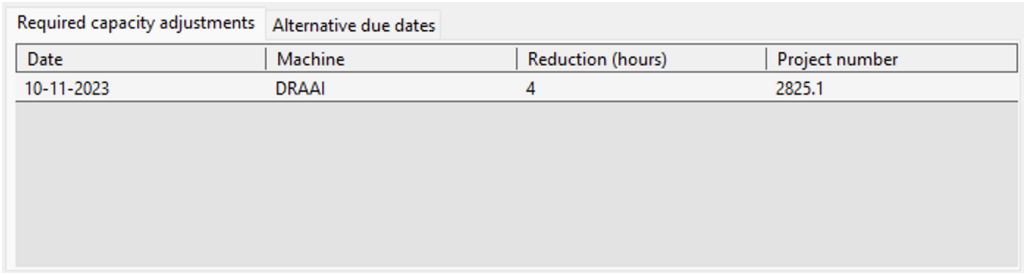
### 4.4.2 Settings

Figure 4.10 shows the various settings that would be available in the tool. The following list summarizes what these do and when or why you might want to use them.

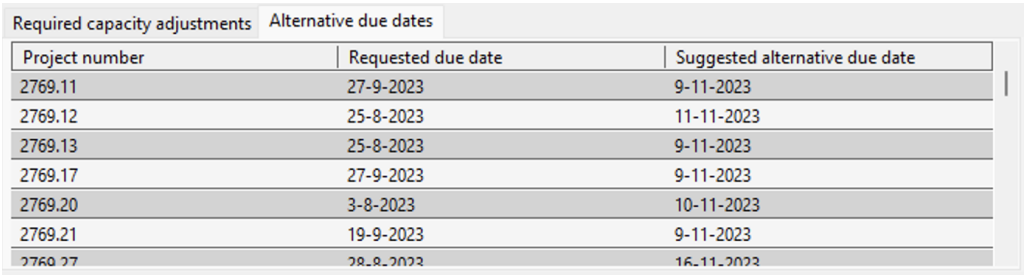
**Processing can start today.** This is pretty self explanatory, check this box if it's early enough in the day that processing can still start today. If left unchecked the earliest date an operation can be planned is tomorrow, if checked the earliest date is today.

**Ignore materials.** Allows you to ignore material lead times (while still noting possible violations). You might want to use this to see the effect of disregarding material requirements in a situation in which you could potentially somehow acquire the materials outside of their regular lead times. Maybe this allows earlier delivery of an important order in which case it becomes worth it to you to procure the required materials from a more expensive supplier than can deliver sooner.

**Force critical path.** This allows you to force every operation to be planned according to its critical start date, with the first operation starting today. This only works for must-have orders, and allows you to find exactly where the capacity is missing that




(A) The required capacity adjustments table




(B) The alternative due dates table

FIGURE 4.9: The output tables of the order acceptance tool, with example content

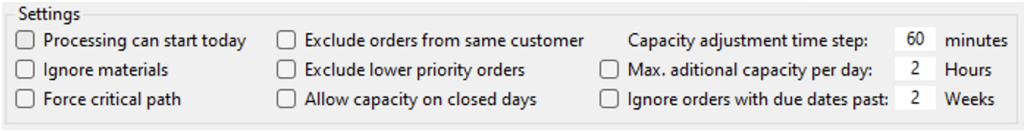



FIGURE 4.10: Settings in the proposed order acceptance tool

is preventing you from delivering the order as soon as possible. Use cases for this might be limited, but might come in handy in some niche scenarios.

**Exclude orders from the same customer.** This allows you to disregard other current orders from the same customer, i.e., you do take into account the machine loads that they take up. This allow you to play within the capacity that you had already previously allotted to this customer. You can then see how this would affect the order's expected due date or potential required capacity adjustment. You might want to use this in case a customer has a rush order that is really important to them. So important that it can have priority over their other orders. Or when you simply want to see what is possible within the customer's "own" hours, so without affecting other customers. This enables you to then further discuss and/or negotiate with the customer about the possibilities.

**Exclude lower priority orders.** Same as the previous except instead of orders from the same customer, you now ignore all other orders with a priority lower than that of the order under evaluation. Another version of this would be to to ignore all orders below a certain chosen level, independent of the level of the order under evaluation. This again allows you to observe the possibilities if you were to ignore the lower priority orders for a minute.

**Allow capacity on closed days.** This is only for the capacity adjustment finding procedure. If this is enables, it allows the procedure to add capacity to otherwise closed

days. Without this enabled, the procedure skips days that have 0 capacity, but with this enabled it is allowed to try fitting the order with added capacity to those days as well.

**Capacity adjustment time step.** The value here determines with what value the capacity adjustment fitting procedure decreases the processing time of the order, before trying to fit it again. This is often not very useful for end users, but is more so there for testing purposes. Further explanation can be found in the previous section on experiments (4.2).

**Max. additional capacity per day.** Much like the name suggests, this value, if enabled, determines the maximum amount of time of capacity per day. The capacity adjustment fitting procedure is then not allowed to go over this amount of extra time per day. Say this is enabled with 2 hours as value, then the procedure will only go up to at most 2 hours of over time per day. This can be useful setting if you want to limit the amount of possible over time hours per day.

**Ignore orders with critical start dates past.** If enabled this ignores all currently planned orders that have critical start dates the chosen number of days past the requested due dates. this allows you to filter out orders that have been pushed forward to keep machines busy, but that don't necessarily have to start until a certain amount of days after the requested due date. Say you use 5, then all orders with a critical start date at least 5 days later than the requested due date of the order under evaluation will be ignored. Again ignoring meaning that their machine loads are not taken into account. This might be very useful in cases where there is a lot of work in the system that has been pushed forward in the schedule.

It is very conceivable that there are a plethora of extra setting/ options that might be interesting to add in the future. We believe these to already be a pretty broad selection of extra option, but more might certainly arise as customers voice their needs and wants after the tool has been introduced to them.

We have showcased/ demonstrated the tool in this form to an actual customer, showing them what they can do with the tool and how they would use it. Their feedback was exceedingly positive on all fronts, i.e., the functionality, the simplicity of use, and the speed with which it returns results.

## 4.5 Integration into existing software (Limis Planner)

There has been some considerable thought and effort put into the integration of the proposed order acceptance tool into the existing software, i.e., Limis Planner. Both in design of the UI as well as its functionality.

First of all, and perhaps most importantly, the proposed tool fully works with the same actual real customer data, straight from the underlying database. When launching the tool, this data is automatically loaded and processed to setup the virtual manufacturing environment of the customer. Aside of just taking all this required data from the database, it also communicates with the current order acceptance tool in Limis Start.

In the current tool the orders that are placed in the middle table are the orders that need to be evaluated. The new (proposed) tool actually reads out this data and adds those

orders to its own "orders to evaluate" table. This means that you can actually add orders to be evaluated by the new tool, by moving them in the current tool interface in Limis Start.

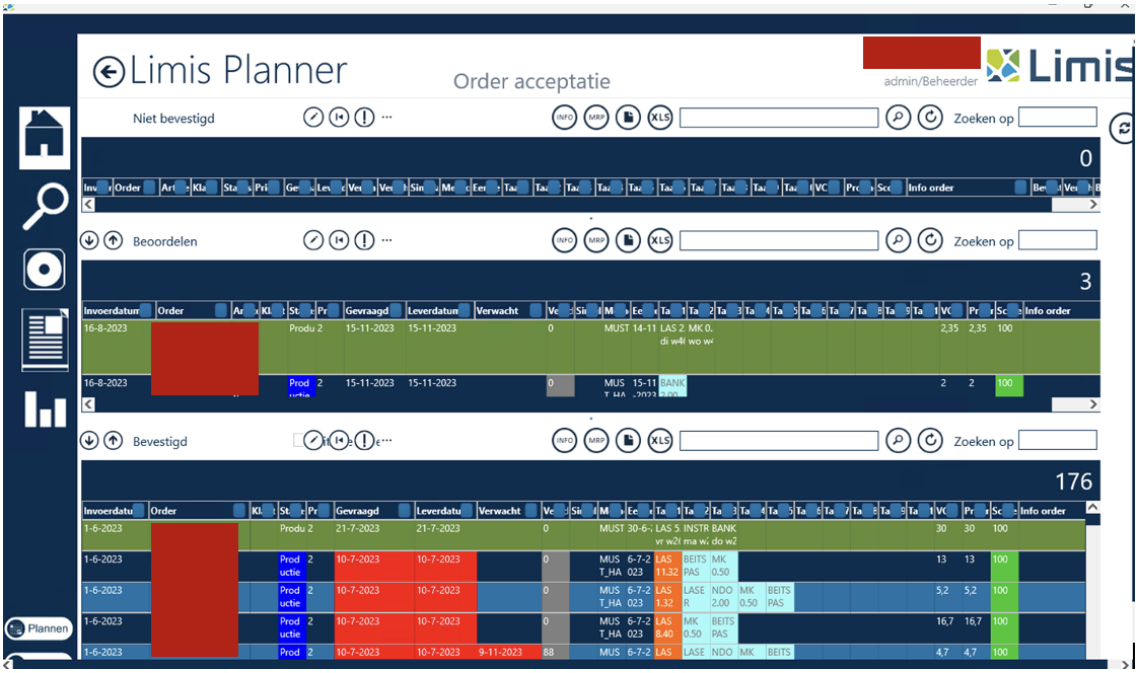



FIGURE 4.11: Screenshot of the current order acceptance tool within Limis Planner

Figure 4.11 shows the interface of the current order acceptance tool. The bottom table contains all the orders currently in the system that have not yet been completed/ finished. If you want to reevaluate one of these orders, you can use the up arrow button to bring it to the middle table. Doing so will also actually add it to the new order acceptance tool, allowing you to run the new order acceptance methods on it. So the input data to the tool has been fully integrated with the existing software.

Furthermore, the alternative due dates found by the new tool, are also exported back to the database and thereby Limis Start. Order/ project records in the database have 5 spare cells, called "Vrije tekst 1,...,5". Not a single row has any data entered in the 5th one, allowing us to for now write our found alternative due dates back to this column. Doing so will then also make it show up in the records in Limis Start itself. So aside of having to startup the tool separately (from outside Limis Start), it is essentially already integrated within the existing software to some extend.

To further integrate the tool, it would need to be able to launch from within Limis Start, which should not be hard to do at all. Furthermore, the scheduler could be adjusted to optionally take this generated alternative due date as input parameter. Finally, the style of the UI could be made to resemble the existing UI more. These further integration efforts were not worked out further as they were determined to be outside of the scope of this assignment. In fact, we've already gotten way more work done on the integration front than initially expected.

# Chapter 5

# Validation

In building the proposed new order acceptance tool we've put great care and thought into validation (and verification) at each step of the way, to ensure a valid end product. The validation process included thorough testing, feedback loops with stakeholders, and continuous improvement to guarantee that the new order acceptance tool and its methods are as valid as possible. In this section we go over the validation of each part.

## 5.1 Loading the data

Before we can run the order acceptance procedure, we first need to load all the required data from the database, i.e., factory data, machine data, order data, etc. To do so we use the standard system.data.SqlClient package in our c# code in Microsoft visual studio. This allows us to establish a connection to the database and perform SQL commands on it. We've put great care in using the right variables from the right table as well as correctly interpreting their unit of measurement (hours, minutes, number, etc.). Doing so took quite a while and required multiple discussions with stakeholders within Limis. Notably, this approach allows us to setup everything correctly independent of which customer's data is being used, as all customer databases have been standardized for use within Limis Planner already

Of course we had to make sure these were all correct as otherwise it would be impossible to make a valid tool. Therefore we created a table in which we registered for each variable used in each function, which database variable we used, from which table, how we interpret its value, and its intended purpose. This overview allowed us to easily track and correct possible mistakes throughout the process of building the tool as well as upon its completion. This all in an effort to make absolutely sure that everything was being used correctly. An added benefit to this is that this enables Limis to understand the source code better and allows them to keep track of things for future updates and integration into their existing software solutions.

The data had to be taken from 7 different tables in the database. To avoid corruption of the data we first load/ copy the data into the memory and only perform operations to the then locally stored data. The following is an example of how we did this:

```
1 wpData = DbConnect.loadData(DB, "select * from a_werkplek");
```

In which wpData is defined as a DataRowCollection that we are now free to perform whatever operation to, without affecting the data in the database. The table with the

used variables can be found in Appendix A. Note that the actual info on the database variables and tables have been replaced due to possibly being sensitive data.

### 5.1.1 Machines

Things we've had to consider in regards to the machines(groups) are that the capacity should be date specific. That way we take into account for possible deviations from the standard daily capacities, stemming from for example holidays or employee absence. Connected to this is that a machine can be an external machine, in which case we assume infinite capacity and a standard throughput time. The occupied machine capacity by currently scheduled orders was taken by looping over all scheduled orders in the system and summing up the machine loads on their respective dates. The capacity left available on a date can then be found by simply subtracting this from the max capacity on that date.

Since we're working with machine groups potentially existing of more than one machine per group, we had to take into account a maximum hours per day for an operation (on a single machine). If for example a group consists of 5 machines and there is a total of 20 hours left available, this does not mean we can perform 20 consecutive hours of operations. The longest consecutive operating hours would for example be 8, as each individual machine is in this example only operated for 8 hours on this day (9 to 5). Luckily this "standard machine roster" is available for each day of the week for each machine. We take this into account in practice by saying that the available capacity for an operation when checking a date, is equal to the minimum of the available capacity and the day's maximum single machine capacity. So in the example above it would be 8 and not 20.

### 5.1.2 Orders and planned capacity

When it comes to the orders, we first off had to make sure that we setup the project/ order structure correctly. Each Project consists of orders which consist of operations, which can in turn require materials. Furthermore, orders can also have sub-orders in case of an assembly type routing. the assembly type routing complicate things and require the use of recursion to set up and loop through an order correctly.

All the orders in the system are either marked as an "what-if" order or a a regular order. If an order is a regular order, we're only concerned about the machine capacity its operations take up on their respective dates. As these are the orders that are readily scheduled. If an order is a "what-if" order however, it is not yet scheduled and it is an order that we need to evaluate in our order acceptance procedure. So in that case we have to load all its relevant data (project number, order structure, operations, materials, etc.), and setup the the order in its correct structure (routing).

All the loaded machines, orders, and resulting capacity profiles have extensively been tested to be correct and accurate. This was done throughout each step of the way, to ensure that everything was still functioning correctly and validly. In order to do so we have, aside of meticulously debugging the code, written testing / displaying functions for all procedures, so that we could check if they behaved as expected. The following code is an example of a function that displays an order/ project in its correct structure, including its operations and info like its throughput time, planned date, etc.

```
1 public string DisplayOrderRouting(Project project)
2 {
```

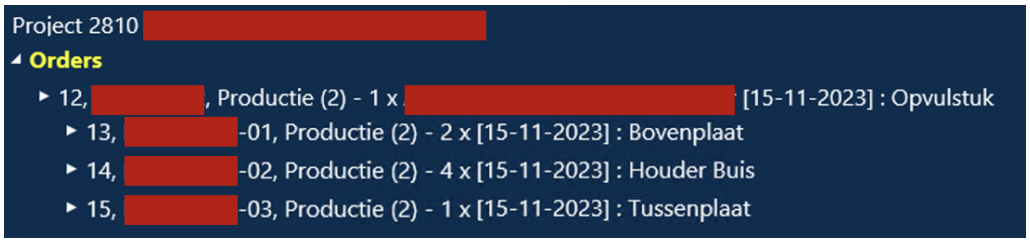
```
    string text = "Project " + project.ProjectNumber + "\r\n";
    return  text + DisplayNextLevelOrders(project.Orders.First().
  Value, 0);
}

public string DisplayNextLevelOrders(Order order, int
  indentationLevel)
{
    string text = new string('\t', indentationLevel);
    text += "- [Order " + order.OrderNumber + "]: " +
       order.PlannedFinishDate.ToShortDateString()
        + " " + order.OrderThroughputTime() + ".\r\n";

    foreach (Order subOrder in order.SubOrders.Values)
    {
        text += DisplayNextLevelOrders(subOrder, indentationLevel
  + 1);
    }
    return text;
}
```

Figure 5.4 shows an example output of this display function (A), and the same order displayed directly in Limis Planner (B). Looking at the two figures we can see that in they match up correctly. Concludingly, we can confidently say that all the required input


(A) Example order as out of display function.




(B) The same order as displayed directly in Limis Planner

FIGURE 5.1: An example of an order as displayed in the output of the the order display function (a) and as displayed in Limis Planner (b)

data is correctly taken from the database and validly represents the situation at hand at the customer in question. That is, it is as valid as the the rest of Limis Planner as it is correctly using the same data in the same manner.

## 5.2 Procedures

Now that we know that our input data is valid, we can discuss the validity of the procedures responsible for handling and transforming the input data into an output. This includes various main procedures such as the finite loading functions as well as smaller helper functions that are used along the way.

### 5.2.1 The initial check (backward loading procedure)

First of all, we have the initial check, i.e., the backward finite loading procedure used to check if an order fits before the requested due date. This procedure starts at the last operation in the order and starts at the requested due date. It then goes backward in time trying to find a day on which the required machine has enough available capacity to process the operation. It does so until one of several stopping criteria is reached. The first one being going past the current date. It goes without saying that there is no need to look at dates that lie in the past. Therefore if we surpass the current date we can conclude that the order doesn't fit before the requested due date.

The second one is, in case the customer has their materials included in Limis planner, if the search data surpasses the current date + the longest lead time of the material needed for the operation. If this date is surpassed we assume we can not obtain the required materials in time, and therefore we can stop the procedure and conclude that the order does not fit. To that end we first find the number of days between the current date and the day we're looking at. We then compare this to the required lead time. To find the required lead time we have a function that is part of the operation class, that simply finds the maximum of the known lead times for each of the materials required for the operation. Code snippet 1 in Appendix B shows both the checks on stopping criteria as well as the function for finding the longest lead time.

The next thing we have to consider is whether the operation can be handled in a single day, or requires multiple days as these require a different approach. For so called single day operations we just need to check if a machine on that specific day has more available capacity than operations processing time. For what we call a multi-day operation however we need to find a day that not only has available capacity to start the operation, but its following consecutive days need to have at least a full single machine day worth of capacity available up until its sum surpasses the remaining total processing time. This because we assume all operation to be uninterruptible. A full single machine day here means the day's maximum single machine capacity as explained earlier, e.g., if machines operate between 9 and 5, their are at most 8 consecutive hours available on a single machine that day.

Figure 5.2 illustrates an example of such a multi-day operation being loaded. In this example we see that we do not only need the 3 hours capacity on day 6, but also two full days of 8 hours on day 7 and 8, as well as roughly 3.5 hours on day 9. Code snippet 2 in Appendix B shows the so called multi-day checking procedure, that does exactly that. We see that the procedure checks if the days after the first day have at least a full day available until the total capacity is greater than the operation processing time. Note that it takes into account closed days and correctly adjusts for them. Else it would incorrectly reject the operation if a closed day (weekend, holiday, etc.) during the period it is checking. In fact an order that takes longer than 5 days would become impossible to fit, assuming we're dealing with a machine that is closed over the weekend.

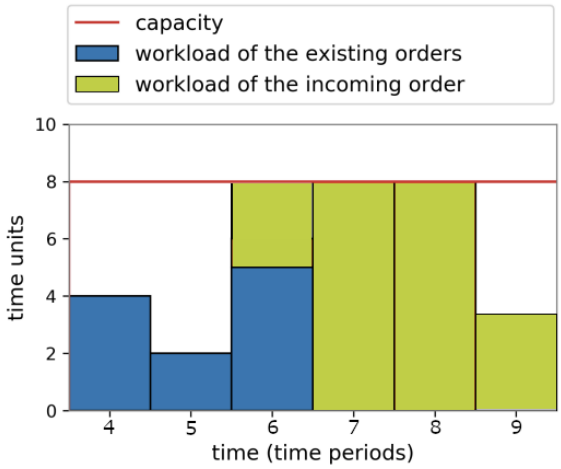



FIGURE 5.2: Example of a multi-day operation being loaded on a machine

subsequently, when setting the search start date for the next operation in the order we have to take into account the (expected) throughput time of the next operation. We can of course not schedule an operation that takes 5 days, a day before the scheduled date of the operation that is next in line in the routing. Therefore, if the throughput time of the operation we can start searching on the same date, but otherwise we gotta subtract a number of days equal to the throughput time. This is handled by the following code:

```
if (operation.ThroughputTime == 0)
{
    operation.StartDate = previousPlannedDate;
}
else
{
    operation.StartDate = previousPlannedDate.AddDays(-operation.
  ThroughputTime);
}
```

Before the backward loading procedure starts, there is a check on the requested due date of the order. If this due date lies in the past, the order obviously doesn't fit, as we can not go back in time. Even more so, if the requested due date does not lie further out from now than it's critical path length, we also know that the order will never fit. This because the critical path length is simply the minimal number of days needed to finish the entire order, from beginning to end. So even with infinite capacity, if you can process every operation one right after another, you will always need at least this number of days. So again of course, we can not start a new operation on a past date. Therefore, an order which requested due date is not later than the current date plus the length of its critical path will never fit. As a result we do not have to run the backward loading procedure, as we already know the order doesn't fit before the requested date.

In order to be able to check this, we have to first determine the critical path of an order. If the order has a string type routing, we can simply sum up all the throughput times of the individual operations, to find the critical path. If however the order has an assembly type routing, i.e., the order has sub orders that have to be finished before it's own operations can start, it is not as simple. In this case, we employ a more intricate algorithm that

considers the dependencies and sequencing of sub-orders, ensuring an accurate assessment of the critical path. To do so we have to to find the critical path length for each individual order that has no suborders. Then we move up a level and find the critical path length for those orders and add onto that the maximum of the critical path lengths of its suborders. This repeats until the critical path length is found for the top level order, i.e., the order that all the other orders fall under. Figure 5.3 shows an explanation of this for an example order with 3 levels of depth.

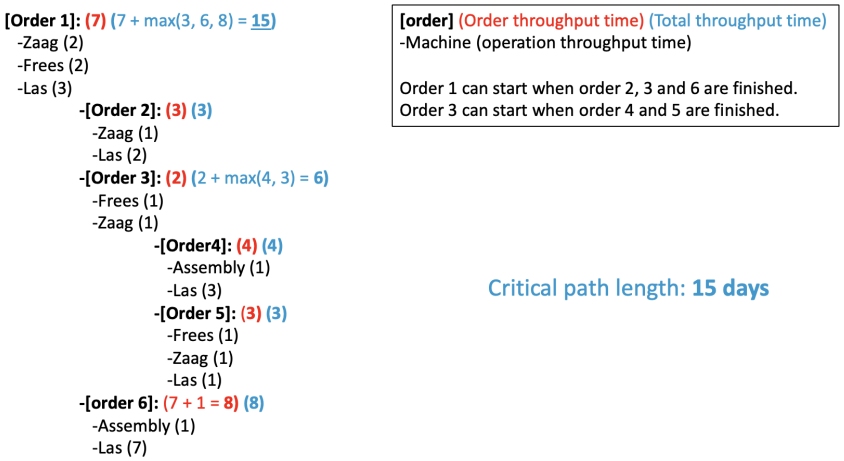



FIGURE 5.3: Example of how to find the critical path length for an order with an assembly type routing

Code snippet 3 in Appendix B shows the procedure used to find the critical path length of an order. This was again meticulously tested to function correctly.

### 5.2.2 Due date assignment (forward loading procedure)

For the forward loading procedure used in the due date assignment portion of the order acceptance approach, we run into the same issues. One notable difference with the backward loading procedure is the search start date. For the first operation under evaluation we do not use the due date but instead use the current date (or tomorrow's date, if its too late in the day to start today). For subsequent operations we instead use the previous planned finish date, i.e., the previous operation's scheduled date + its throughput time. So the start date here depends on the throughput time of the previous order instead of that the current order, unlike in the backward loading procedure.

Just like in backward loading we take into account the closed days when adding this throughput time. To find this next starting date we use the date finding procedure show in the code snippet below. We essentially keep on adding days, counting up their available capacity, until we surpass the total processing time.

```
1 public DateTime findDateFwd(DateTime startDate, Machine machine,
      Operation op)
2 {
3     double load = op.ProcessingTime;
4     int dayCount = 0;
5     while (load > 0)
6     {
```

```
7           double cap = Math.Min(machine.DayCap[startDate.AddDays(
    dayCount).DayOfWeek],
8                   Math.Max(0, machine.CapacityData[startDate.AddDays(
    dayCount).Date].AvailableCapacity));
9           load -= cap;
10          dayCount++;
11      }
12      return startDate.AddDays(dayCount);
13  }
```

In case of a multi-day operation we move ahead in time until the total processing time has been reached, at each day taking the minimum of the available machine capacity and a full single machine day's worth of capacity. As you can see we just like in all the other multi-day procedures also take the maximum of 0 and the available capacity. This is so that in case of possible over scheduling or data corruption we do not run into any problems with possible negative capacity values.

### 5.2.3 Capacity adjustment (fitting procedure)

The first thing that happens after we find that the order doesn't fit, is that the procedure finds the operation with the longest throughput time. In order to do so we again go over the order recursively, and use Linq to add the operations that have not yet been reduced to 0 processing time to the list. We then loop over the list comparing the throughput time to the current longest. We calculated throughput time as the time between finishing the previous order and finishing the current order, minus the number of closed days in this period. Code snippet 4 in Appendix B shows the procedure that we have written, that takes care of this for us.

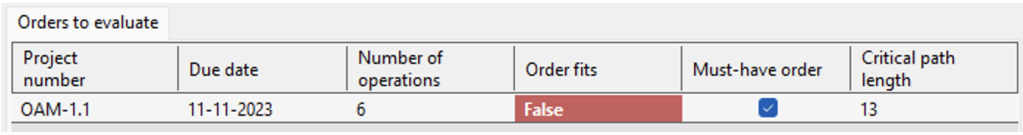
After we've found the operation with the longest throughput time, we want to decrease it's processing time before running the backward loading procedure again. Furthermore, we need to keep track of the amount of time we reduce the order by and on which machine and date we end up doing this as shown in code snippet 5 in Appendix B.

Finally, we need to go over all the reductions that we have found and add them to the reductions table so that they are correctly displayed in the tool. After adding the reduction to the project, we set the processing time of the operation back to its original, and its reduction back to 0. We do this so that we can perform multiple runs without having to restart the tool. Without this, the reductions keep adding up and the processing times get all messed up. The used procedure can once again be found in Appendix B, in code snippet 6.

## 5.3 The output

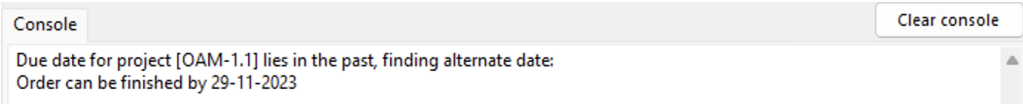
Now that we've seen that the procedures are valid, we can confirm that their output is valid as well. We do so by checking the outcomes to be in line with our expected outcome for known input. Furthermore, we confirm that the tool plans everything as expected, by inspecting the actual machine capacity graphs in Limis Start, and use them to validate each operations planned date. Here we'll give one example for each of the procedures, based on actual data and some actual orders of company x. Note that in these examples The current date is 5 November 2023, which is a Sunday.

First of all, when we enter a requested due date that lies in the past, or that is not further out than the current date + its critical path length, it should return False, as the order can never fit. Figure 5.4 shows this for the example order. We see that the entry in the "Order fits" column correctly shows False. Furthermore, the output reads, "requested date lies in the past, finding alternative due date."


Orders to evaluate

| Project number | Due date | Number of operations | Order fits | Must-have order | Critical path length |
|---|---|---|---|---|---|
| OAM-1.1 | 11-11-2023 | 6 | False | ☑ | 13 |

(A) Example order.


Console

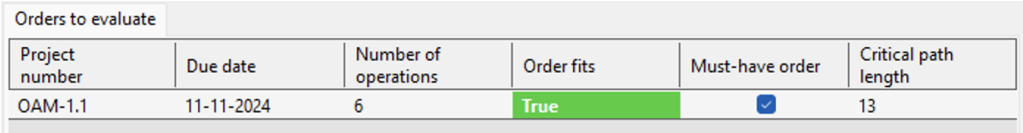
Clear console

Due date for project [OAM-1.1] lies in the past, finding alternate date:
Order can be finished by 29-11-2023

(B) The output of running the procedure for this order

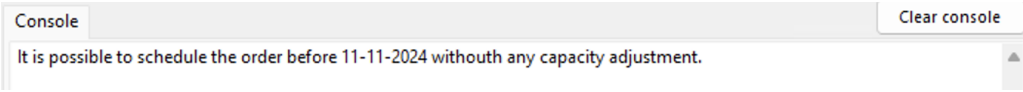
FIGURE 5.4: An example of an order with a due date earlier than the current date + the length of its critical path.

If we instead enter a due date that is really far out, we expect the result to be true, as the order is a rather simple order with a critical path length of only 13 days. Figure 5.5 shows an example for the same order as before but this time with a due date near the end of 2024. We see that Order fits column now reads True, and the output says "It is possible to schedule the order before 11-11-2024 without any capacity adjustments".


Orders to evaluate

| Project number | Due date | Number of operations | Order fits | Must-have order | Critical path length |
|---|---|---|---|---|---|
| OAM-1.1 | 11-11-2024 | 6 | True | ☑ | 13 |

(A) Example order.


Console

Clear console

It is possible to schedule the order before 11-11-2024 withouth any capacity adjustment.

(B) The output of running the procedure for this order

FIGURE 5.5: An example of an order with a due date that is relatively far out, to the point it should have no problem fitting the order

We saw in the first example that it used the forward loading to find the first feasible due date. Now lets use Limis start to confirm that this is indeed correct. By clicking on the number of operations, the tool displays exactly when each operation in the order is planned and what its (minimal) throughput time is. We can check for each operation if this is indeed correctly planned. We start for the order planned on the suggested date. Figure 5.6 shows when the operations are planned according to the OA tool. We see that the project consists of two orders, 1 and 2, each consisting of 3 operations. Order 2 is a sub order of order 1 and therefore needs to be finished before order 1 can start. We also see that it correctly finds the delivery date of 29-11-2023 for order 1.

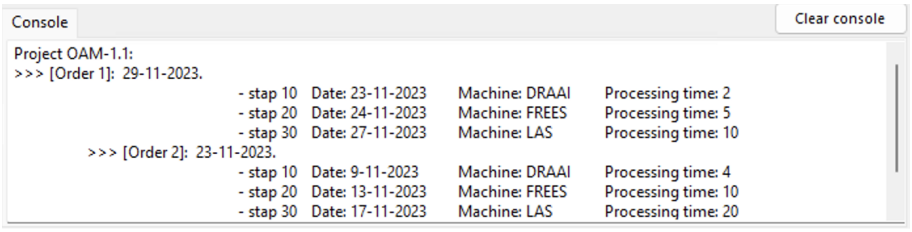



FIGURE 5.6: Structure of the example order, including planned date for each operation.

Now to confirm this with Limis Planner, we look at the respective machine graphs to see if these operations can indeed be fitted on these dates, given that it is Wednesday 8 November 2023 at the time of writing this. Starting with the first step of order 2 (stap 10), the first date this order can be planned is November 9, as we have not checked the order can start today checkbox. We will now for each step show the capacity graph of its respective machine.

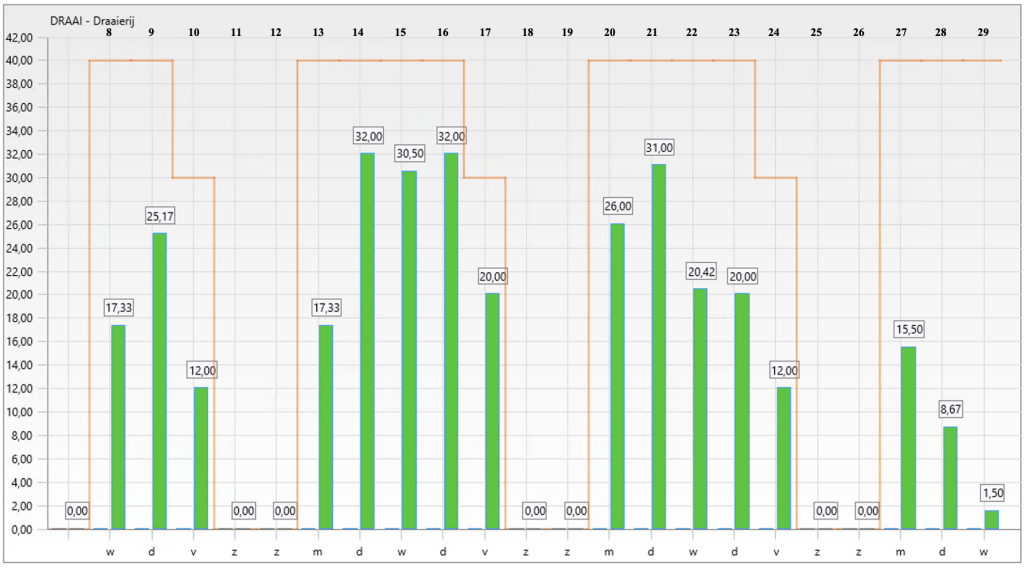



FIGURE 5.7: Machine capacity graph for Order 2 stap 10. (DRAAI)

We see in figure 5.7 that we have 8 hours available on machine DRAAI on the 9th of November, which is plenty to fit the 4 hours that we need for stap 10.

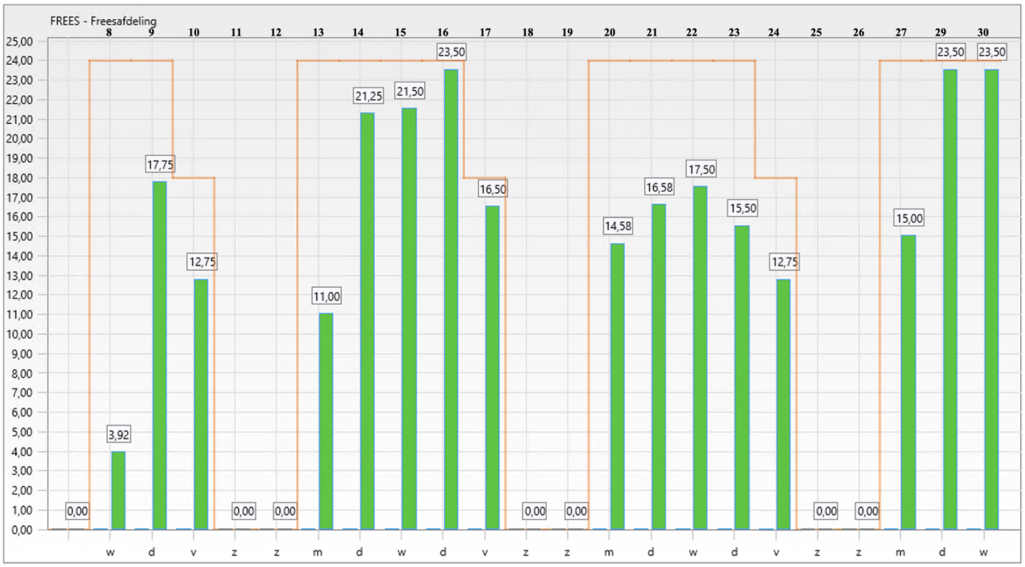



FIGURE 5.8: Machine capacity graph for Order 2 stap 20. (Frees)

We see in figure 5.8 that we have 13 hours available on machine FREES on the 13th of November, however a single machine has only 8 hours this day. So taking of 8 hours leaves us with 2 hours remaining, which we see fit on the next day (the 14th) as it has 2.75 hours left available. Therefore, we can indeed schedule order 2 - stap 20 on the 13th.

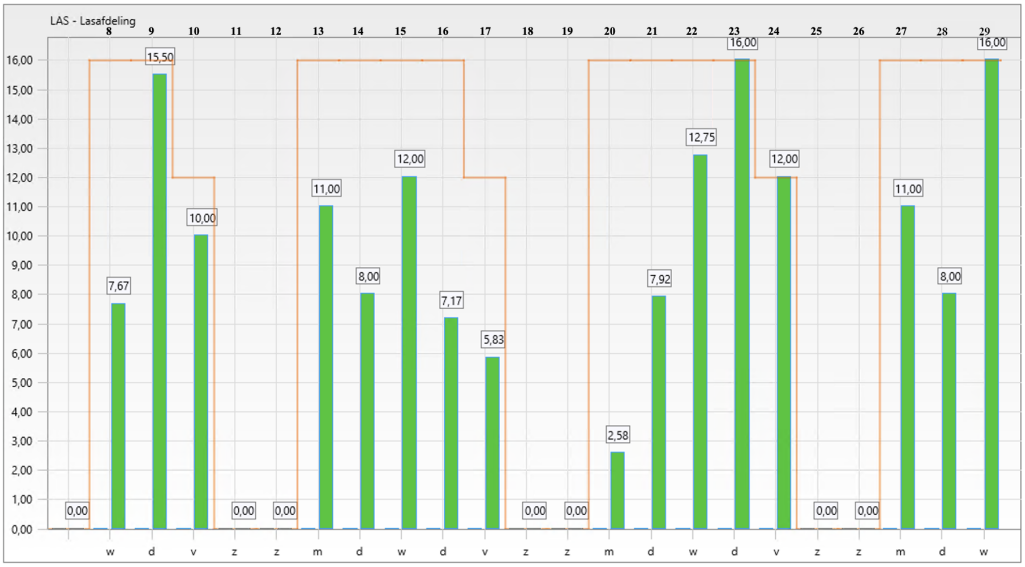



FIGURE 5.9: Machine capacity graph for Order 2 stap 30. (LAS)

Order 2 - stap 30 requires 20 hours on LAS, with again a maximum of 8 hours per day. The tool plans this operation on November 17th. If we look at figure 5.9, we see that we can schedule 6.17 hours on Friday the 17th, the maximum of 8 on Monday the 20th and then the remaining 5.83 on Tuesday the 21st. Therefore, we can indeed schedule order 2 - stap 30 on the 17th. Note that The 15th and 16th don't work as Friday does not have a full 8 hours available, meaning we can not assume that the order can be run uninterrupted. Similarly when planning the operation on the 17th, we need at least a full 8 hours on Monday the 20Th for this same reason, which indeed we do have since $13.42 > 8$.

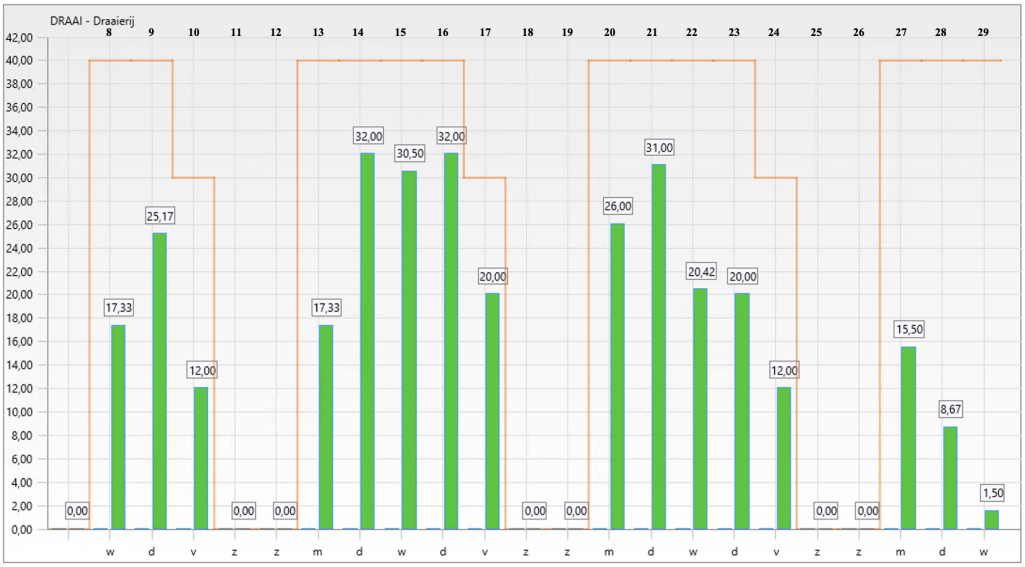



FIGURE 5.10: Machine capacity graph for Order 1 stap 10. (DRAAI)

Having finished order 2 by the 22nd, we can start working on order 1 - stap 10 on the 23rd. Looking at figure 5.10, we see we have plenty of hours available on this day, as we only need 2.

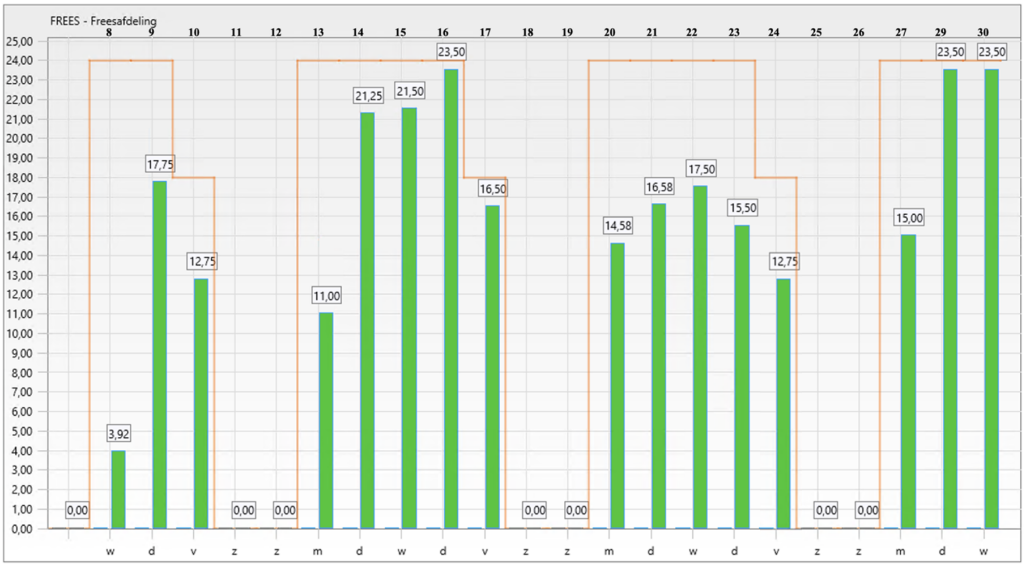



FIGURE 5.11: Machine capacity graph for Order 1 stap 20. (FREES)

Order 1 - stap 20 requires 5 hours on FREES, and can start on the 24th. If we look at figure 5.11, we see that we have 5.25 hours available on this day, just barely enough to schedule stap 20.

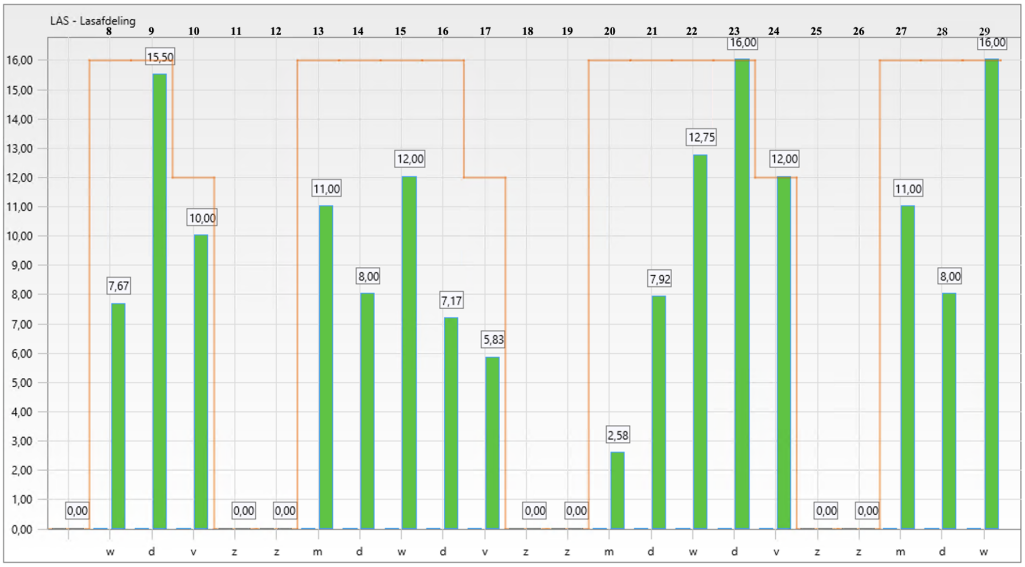



FIGURE 5.12: Machine capacity graph for Order 1 stap 30. (LAS)

Finally, Order 1 - stap 30 requires 10 hours on LAS again. On the first possible day, Monday the 27th, we have 5 hours available, and on the next day we have 8 hours available, meaning we can schedule the order here. So finishing the final operation of the order on the 28th, we can deliver the next day, so the 29th. Therefore, like the tool indicates, we can conclude that the order fits before/ can be delivered on November 29th.

Now if we instead try to fit the order on a day earlier, we expect the outcome to then be false, given that it is a best-fit order and so we're not adjusting capacity. Figure 5.13 shows that the outcome of the procedure is then indeed false. This is perfectly in line with our expectation. As shown in the previous example this also makes perfect sense, as there the order exactly fitted before the due date, leaving no room to expedite the order by a day.

Orders to evaluate

| Project number | Due date | Number of operations | Order fits | Must-have order | Critical path length |
|---|---|---|---|---|---|
| OAM-1.1 | 28-11-2023 | 6 | False | ☑ | 13 |

FIGURE 5.13: Result of trying the procedure with a day earlier than in the previous example.

## 5.4 Conclusion

We've seen that the input data is loaded and processed correctly. We've also seen that the procedures and their logic function/ are correct. But above all we've seen that the output of the tool matches the expected output for a given input (order, date etc.). Not only does the proposed tool do what it is supposed to do according to the conceptual model, we've also shown through actual real examples that it adheres to reality. At least as far as the data within Limis Planner is correct. Since this is the target environment within which the tool should operate, once integrated completely, we are not concerned beyond that. We can therefore confidently say that the proposed approach and the resulting Proposed

tool are indeed valid, i.e., it correctly does everything we want it to do.

# Chapter 6

# Numerical study

In this chapter we present the design and results of the experiments that were part of the performed numerical study. This includes the performance of the proposed solution, some parameter setting, and testing classification models. Section 6.1 explains the data used in the tests. Section 6.2 describes the design and goals of the conducted experiments. Section 6.3 shows and discusses the results of these experiments. Finally, Section 6.4 finishes the chapter with a brief conclusion.

## 6.1 Experiment Data

The data used in the tests pertains to three different sized customers, i.e., a small, a medium size, and a large customer. These sizes are relative to the number of machines the customer has, and number of (active) orders/ operations in their system. Table 6.1 shows the specifics for the 3 different customers whose data we use. As the proposed solution is

Table 6.1: Details of customers whose data the experiments are based on.

| | Small | Medium | Large |
|---|---|---|---|
| Number of machines | 11 | 70 | 287 |
| Number of operations | 42 | 5694 | 52036 |

going to function inside / as part of Limis Planner, this is the environment in which it has to perform. Meaning that Limis Planner forms the baseline for the desired quality, i.e., we assume the outcomes of the scheduler in Limis Planner to be correct. This also makes sense for the eventual users, as they already use, and trust, Limis Planner to schedule all of their orders. Furthermore, the goal of this study was to design a new order acceptance tool to replace the old one, i.e., we in basis want to replicate the performance of the old tool but find the solution (much) quicker. We therefore compare the performance of the new tool/method to that of the old tool.

When testing the proposed tool (and comparing it to the old one) it is of course important that the cases/ scenarios are exactly the same between the test and the benchmark/ reference. In our case meaning that we run the exact same case in both Limis Planner and in our proposed tool. Luckily this is easily achieved, as we've built the tool to use the data directly from the customer's database, in the same exact way Limis Planner does. This

means that when you startup the tool, it loads in the machines, orders, etc., to correctly recreate the situation at hand at the customer. In fact, as explained in Section 5.4, if you insert an order to evaluate in Limis Planner, it will also appear exactly the same in the proposed tool.

Note that these customer databases are of course not the live production databases, but rather a clone/copy used specifically for testing. This because we of course do not want to in anyway interfere with operations. At the same time, the fact that they are one time copies, ensures that the data does not (suddenly) change over time, but rather stays the same, throughout time and experiments.

It is also important to note that the scenarios/cases in the experiments are each based upon the database / schedule as it was on the specific date of creating the scenario. That is it takes the schedule (and resulting machine capacities) as it was on that day. This means that all experiments must be (and were) performed on the same day.

## 6.2 Experiments

In Section 6.2.1 we explain how we conduct experiments to determine the general performance of the proposed solution. In Section 6.2.2 we explain the experiments involved in parameter setting for the iterative capacity control approach. Finally, in Section 6.2.3 we explain how we experiment with classification models as an alternative way to determine if an order fits or not.

### 6.2.1 Performance of the proposed solution

The goal of these experiments is, as the name suggests, to determine the general performance of the proposed solution. Since the main goal of this study was to design a quick order acceptance method/ tool, the performance consists of two aspects, how quick is the method (computation time), and how accurate / correct are its found results (solution quality). We have seen in Section 5.3 that the method finds valid results, however, we would ideally test the actual tool against the results found by the current tool/ the scheduler. We planned on doing so by taking a case (order with a due date) and having it judged by both our proposed tool and the existing one (based on the scheduler). The idea was then that if our tool says the order does fit, the scheduler should be able to schedule the order on a date equal to or earlier than the requested due date. Vice versa, if our tool says the order doesn't fit, the scheduler should produce a scheduled date later than the the requested date.

Starting these experiments we quickly realized that for any case where the answer was not immediately obvious, the results of new tool deviated from the scheduler's results significantly, i.e., it found/ accepted a much earlier date than the scheduler would for every case. What we mean here with "cases in which the answer was not immediately obvious" is any case for which none of the following holds true:

**Due date lies in the past:** in case the the requested due date lies in the past, both methods obviously return that the order does not fit, as this is simply not possible.

**Due date is not further out than the optimal critical path length:** likewise, in case the the requested due date does not lie further out than the current date + the length of the optimal critical path, both methods obviously return that the order does not fit, as it is not possible to process the entire order before this date.

**The system is empty:** if the system is empty, that is, there are no other orders that need to be processed other than the one we're evaluating, both systems simply accept anything after the current date + the length of the optimal critical path, and reject ever case with an earlier due date. This because if the system is empty both methods simply load/ schedule each operation right after another, as there is simply nothing that delays an operation.

**The requested due date is far out:** if the requested due date is far enough out, it will simply always be accepted in both methods. Say we have an order with a throughput time of a week, and we want to evaluated this order with a due date a year from now, it will always fit. Both because our data doesn't go that far out (we only have data for at most a couple of months in the future), and because no Limis user is fully planned to maximum capacity that far out, so there will always be a spot where the order can fit.

So if none of these cases apply, the proposed method consistently finds/ accepts a due date much earlier than the one found by the existing tool/ scheduler. The reasons for this are pretty obvious however, and have everything to do with the limitations/ simplifications that we have to deal with.

There are a myriad of reasons, why at this time these two results do not match each other. Most importantly, and also perhaps the biggest limitation to this study, is that The new method can only work with the capacity of a machine group, and not the capacities of each individual machine, unlike the scheduler. This is because, outside of the scheduling process, capacity is only stored on machine group level, rather than individual machine level. If it isn't obvious why this means that the proposed method finds a earlier due date than the scheduler because of this reason, imagine the following: You need to schedule an operation that takes 6 hours uninterrupted to process on a machine of type X and you have 3 of these machines. Each machine operates 8 hours per day Monday to Friday. Now imagine an arbitrary week in which other operations have been scheduled on the machines and table 6.2 below shows the planned workload for each day and table 6.3 shows the capacity left available on each day.

Table 6.2: Example planned workloads for machines of type X

| Day | MON | TUE | WED | THU | FRI |
|---|---|---|---|---|---|
| Machine 1 | 5 | 7 | 8 | 8 | 1 |
| Machine 2 | 7 | 8 | 8 | 8 | 6 |
| Machine 3 | 6 | 8 | 8 | 8 | 6 |
| Combined | 18 | 23 | 24 | 24 | 13 |

Table 6.3: Example capacity left available for machines of type X

| Day | MON | TUE | WED | THU | FRI |
|---|---|---|---|---|---|
| Machine 1 | 3 | 1 | 0 | 0 | 7 |
| Machine 2 | 1 | 0 | 0 | 0 | 2 |
| Machine 3 | 2 | 0 | 0 | 0 | 2 |
| Combined | 6 | 1 | 0 | 0 | 11 |

Now looking at table 6.3 we see that the first day for which a single machine has 6 or more hours available is machine 1 on Friday, meaning that the first day the scheduler could schedule the operation is Friday. The new proposed method however works with combined capacity, meaning that it would say that the operation fits on Monday, since all machines combined have 6 hours of capacity left available on Monday. Now imagine an order consisting of multiple operations in which a similar situation can occur and it quickly becomes apparent why the use of capacity on a machine group level, rather than an individual level would result in a significantly earlier suggested date on which the order would fit.

Furthermore, the new method does not yet include man capacity for example, furthermore it can not change the existing schedule. Finally, the scheduler does not necessarily always find the optimal schedule, meaning that even though the new method finds a possibility for the new order to fit on top of the existing schedule, it does not mean that the scheduler is actually able to find this solution. It is going to require additional effort to make the new tool and the scheduler work together so that they can come to comparable results. Effort that was not feasible to fit within the time restrictions of this study.

Of course we could have cherry picked our test cases, almost exclusively picking cases in which the answer was always going to be obvious and therefore the result correct. We could then have claimed that our method was super accurate, scoring something like a 99% accuracy. However, this would of course have nothing to do with conducting valid research and would have in our opinion invalidated the entire study. Sadly this does however mean that further evaluation of solution quality (in comparison to the existing tool) will have to remain outside of the scope of this study. Limis will have to look into this themselves, after they have changed Limis Planner to record machine capacity on an individual machine level and have adapted the proposed tool accordingly.

Of course we have however, previously asserted (in Section 5.3) that the proposed method finds valid solutions within the assumption of pooled capacity between machines in a machine group. We therefore focus on the performance with regards to the required computation time to come to a solution, in these numerical experiments.

The comparison that we make with the existing order acceptance tool is on required computation time. Since the old method (tool) simply runs the full (detailed) scheduling process twice, first without, and then with the the new orders that it wants to evaluate, we know that this method takes twice as long as running the scheduler. The run time of the scheduler is something that Limis tracks for each of its users. Despite varying slightly each time, we know an average for each customer. For the new method, we record the current time before running the procedure, and the current time after completing the procedure. The difference between these two times, is then how long the procedure needed to complete, i.e., its computation time. This can be done very accurately in c# by making use of the DateTime.Now() function.

### 6.2.2 Iterative approach parameter setting experiments

The proposed approach, that determines the capacity increase required to make an order fit on a certain date, decreases the processing time of an operation in each iteration. This raises two questions, what operation should be adjusted this iteration, and by how much should its processing time be decreased. We define the amount of time the chosen

operation's processing time is decreased by in each step as the step size. The goal of these experiments is therefore to determine how the procedure should choose an operation to adjust, and which step size to use.

### Selecting an operation to adjust

There might be multiple possible ways of increasing capacity, that allow the order to then fit. Of course we would ideally like to find the most efficient/ least costly way to achieve this. This means that we would like to find a solution that requires an as small as possible capacity increase preferably on the least costly machine. We Therefore experiment with different ways of ordering the operations to be adjusted, to see if there is a way that consistently stands out as better than the others. We test the following four methods:

**Longest throughput time (LTPT)** This chooses the operation with the longest throughput time. Throughput time here being the number of days between the finish date of the previous operation and the finish date of the current order

**Highest order level (HOL)** This chooses the operation that is the highest up in the order structure, and therefore is planned the closest to the due date. The idea is that the higher up you go in the structure the more subsequent orders are affected.

**Highest total machine capacity (HTMC)** This chooses the operation requiring the machine with the highest total availability, as we assume this to be the easiest/ cheapest to find an extra hour on.

**Random (RNDM)** This randomly chooses an operation out of all the possible operations to adjust. We added this for comparison's sake.

We do not directly know the costs of operating a certain machine for an hour, we do however know the total weekly capacity of each machine. We assume that it is harder and/or more expensive to find an extra hour of capacity for a machine that we have less total capacity of, than for a machine that we have more total capacity of. We therefore use the total weekly capacity of a machine as a weight, and since the relation is negative, we multiply the required hours of extra capacity on a machine by 1 over its weight, e.g., $1/80, 1/200$, etc. For example, 1 additional hour with a weight of 80 is more expensive than 2 additional hours with a weight of 200, as $1/80 > 2/200$. We therefore use the weighted total required capacity adjustment as measure of performance (KPI).

In order to perform this experiment we run the the procedure for a multitude of orders. For each order we run the procedure 5 times with, 5 different dates, for each different selection/ ordering method. The five runs use a due date that is 1,2,...,5 days before the first feasible due date without capacity adjustment.

### Step size

We care about the step size, simply because there is a direct trade-off between the needed number of iterations and the overestimation of the required adjustment. Of course more iterations means more computation time, and is therefore undesired. At the same time, a smaller step size means a higher accuracy, as we overestimate the required capacity increase less.

Imagine if you will a hypothetical situation in which an operation's processing time needs

a reduction of 63 minutes for the operation to fit on an earlier date and thereby expedite the order. If we were to work with a step size of one hour, we would find that we need a decrease of 2 hours, in two iterations of the algorithm. If we were to use a step size of five minutes, we would find that we need an increase of 65 minutes after 13 iterations of the algorithm. Figure 6.1 illustrates this example. Of course the first example over estimates the required decrease by a whole lot more than the second example, however it also found the answer in a lot less repetitions of the algorithm.

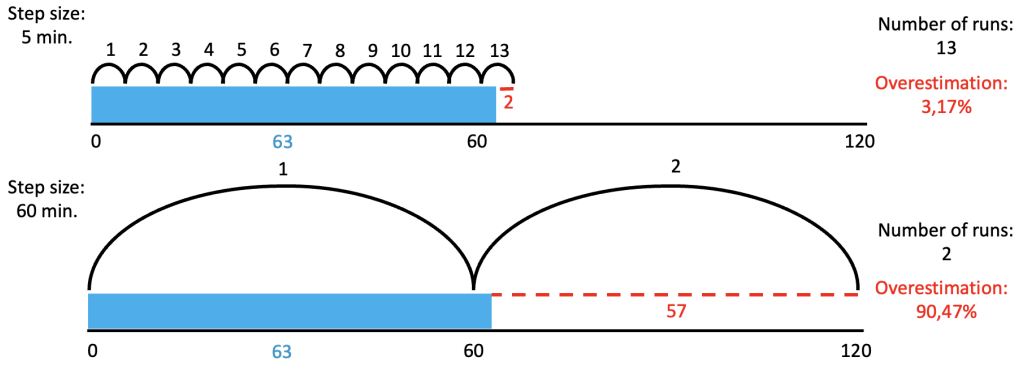



FIGURE 6.1: Illustrated example of different (time) step sizes

Of course you would ideally use the smallest possible time step to find the most accurate result. However, the lower you go the more the computation time explodes. Therefore we have to find a step size that balances the two aspects. To do so we measure the average computation time of the procedure under various different step sizes; 1 hour, 30 minutes, 15 minutes, 10 minutes, 5 minutes, 2 minutes, 1 minute, and 30 seconds.

### 6.2.3 Classification model experiments

As an alternative to the proposed backward finite loading method, we also experiment with the use of classification models. Because Limis was also very interested in exploring the use of AI based methods, we explore the use of classification models to classify a case (order) as fitting or not fitting. The objective of this experiment was to see if we could train a classification model to correctly predict whether or not an order fits. Therefore we again measure an accuracy as an percentage, i.e., how many out of the total cases does the model predict correctly. To this end we use 10-fold cross validation to train and test the models on our data. As explained in Chapter 4, the data belongs to a large amount of cases, evaluated by Limis Planner in order to label them as fitting or not fitting before its respective due date. Both the fitting and testing of the models is done in R. We test the following models:

**Logistics regression** Logistic regression is a regression model used to determine the probability of occurrence of an outcome for an observation. In this case the probability of an order fitting is estimated based on the data given in the training set.

**Decision tree** A decision tree model is, as the name suggests, a tree-like model in which each node represents a test criterion of a feature. Each branch corresponds to a outcome of the test and each leaf node shows the class outcome. In our case this is for each order either 'Fitting' or 'Not fitting'.

**Support Vector Machine (SVM)** Support vector machines are supervised learning models with associated learning algorithms for data classification. These algorithms cre-

ate a line or hyperplane that separate data into classes distinctly classifying all data points in an N-dimensional space. We test support vector machines using three of the most common kernels: linear, polynomial, and radial.

So in total, we test 5 different models, as we test SVM with 3 different types of kernel, i.e., linear, polynomial, and radial.

In order to asses the performance of the models we use four metrics:

**Accuracy** Accuracy simply measures how often the classifier correctly predicts. We can define accuracy as the ratio of the number of correct predictions and the total number of predictions.

**Precision** Precision explains how many of the positive predicted cases actually turned out to be positive. Precision is useful in the cases where False Positive is a higher concern than False Negatives. It is calculates as the true positives over the true positives + the false positives.

**Sensitivity** Sensitivity explains how many of the actual positive cases we were able to predict correctly with our model. Sensitivity is a useful metric in cases where False Negative is of higher concern than False Positive. It is calculated as the true positives over the true positives + the false negatives.

**Specificity** Specificity explains how many of the actual negative cases we were able to predict correctly with our model. Specificity is a useful metric in cases where False positive is of higher concern than False Negative. It is calculated as the true negatives over the true negatives + the false positives.

We use Accuracy as the main performance measure, followed by precision and specificity, as false positives are of higher concern to us than false negatives. A false negative just means we quote a later date for an order, a false positive gets us into trouble with our (customer) agreements. Us in this case being the customer/ Limis planner user.

## 6.3 Experiment results

In this Section we show and discuss the results of the performed experiments. Section 6.2.1 contain the results of the experiments on the general performance and validity of the proposed solution. In Section 6.2.2 we show the results of the experiments involved in parameter setting for the iterative capacity control approach. Finally, in Section 6.2.3 we discuss the results of the experiments with classification models.

### 6.3.1 Performance of the proposed solution experiment results

In this section we discuss the general performance of the proposed order acceptance tool and its methods. In this discussion we draw comparisons to the the current order acceptance tool wherever applicable. Since the main objective of this assignment was the development of a quick tool, the real performance measure here is the computation/ run time. Of course the correctness (accuracy) of the result is also of importance. To that end we have shown in Chapter 5 that the method (and the tool) find valid results, i.e., it correctly determines if an order could theoretically fit before a date. As explained in Section 6.2.1, there is at this moment of time no real use in comparing the results to those of the existing tool, as the found results deviate greatly due to a legion of reasons, the

biggest reason being that we were not able to work with individual machine capacities yet.

Since the computation time required by the current tool, that we deemed to slow, depends greatly on the size of the user/customer, we test the performance of the tool for a small, a medium size, and a large customer.

The computation time is calculated by taking the difference between the time, right before starting the procedure and right after it has finished. We compare the computation time required by the proposed new tool against that of the current (old) tool. For the the results of the order acceptance tool the time is rounded off in hours or minute appropriately. We see that the proposed tool is many orders of magnitude faster than the old method,

TABLE 6.4: Average performance on computation time, of both the old and the newly proposed method

| Method | Small | Medium | Large |
|---|---|---|---|
| current (old) | 5 minutes | 30 minutes | 7 hours |
| Proposed (new) | 0.2934 seconds | 0.8226 seconds | 1.9423 seconds |
| evaluated orders | 2 | 5 | 9 |

especially for larger customers. For the large customers it's really a comparison between several seconds and several hours, which is a truly enormous difference.

Furthermore, we've observed during testing that it is really not so much the size of the customer, but rather the number of orders you want the evaluate at once, and their level of complexity, that determine the computation time. In fact, for single orders of the same complexity (number of operations), there is really no distinguishable difference between the results for the different customer sizes. This makes sense, as the order acceptance procedures only need to check so many operations and machines. All of the computations that might take longer based on the customer size, (loading the machines, orders and their planned capacity, etc.) have essential already occurred behind the scenes. That is to say that all the info is loaded in and processed when starting up the tool, and is not part of the actual order acceptance procedures. Therefore, the required computation time really only scales with the specifics of the order under evaluation and the sheer number of orders to evaluate.

Having said that, we can see that the proposed solution is a massive improvement over the old method, when it comes to computation time, and it isn't even remotely close. Furthermore, The entire procedure is easily capable of running in less than the maximum of 2 minutes that we had set as an objective. Even more so if we're considering the use case in which a customer is being called by their customer with an order acceptance related question regarding a single order.

When initially discussing the objectives of this assignment with the stake holders within Limis, we came to the conclusion that the method doesn't have to be completely accurate/ super precise. The analogy used here was that of a far out weather forecast, i.e., we know roughly what weather its gonna be say 2 weeks from now, but that has a known degree of uncertainty. When it gets much closer to the date we can say with much more certainty that its going to be a specific temperature and whether or not it is going to rain. We agreed that the method would be "good enough" if the proposed solution could indicate roughly

the week in which the order could be delivered. We strongly believe that the proposed solution will be more than capable of doing so, once fully worked out. This will however require some additional effort, to adjust underlying processes and/or further integrate the proposed tool. Mainly the capability to use individual machine capacities rather than that of a group of machines.

### 6.3.2 iterative approach parameter experiment results

#### Choosing an operation to adjust

Before we can decrease the processing time of an operation, we first have to determine what operation we want to adjust. Originally we were just going to use longest throughput time as a measure for selecting the operation to adjust. Since then however, we've come to the realisation that this is not the only option we have, and might not be the best choice either. Therefore we've tested some different options. As a measure of performance we look at the weighted required capacity increase to make the order fit. We test for different due dates, relative to each orders earliest fitting due date, i.e., we test at 1 days prior (-1), two days prior (-2), and 3 or more days prior (-3+).

Table 6.5 shows the results of these experiments. Because these numbers by themselves may not really speak to the imagination, table 6.6 shows the relative performance of each method. These percentages are relative to the average of each number of days prior to the orders respective earliest possible due date.

TABLE 6.5: Operation choice experiment results (average weighted required capacity increase)

| Method | LTPT | HOL | HTMC | RNDM | Average |
|---|---|---|---|---|---|
| -1 day | 0.8614 | 1.0492 | 1.1333 | 0.8958 | 0.9849 |
| -2 days | 2.2136 | 1.3742 | 1.9470 | 1.9290 | 1.8659 |
| -3+ days | 2.9053 | 1.7242 | 2.2970 | 2.6600 | 2.3966 |
| Overall | 1.9934 | 1.3826 | 1.7924 | 1.8283 | 1.7492 |

TABLE 6.6: Relative performance of the tested methods, as a percentage of the average.

| Method | LTPT | HOL | HTMC | RNDM | Average |
|---|---|---|---|---|---|
| -1 day | 86.57% | 105.46% | 113.91% | 94.06% | 100% |
| -2 days | 118.63% | 73.65% | 104.34% | 103.38% | 100% |
| -3+ days | 121.22% | 71.94% | 95.84% | 110.99% | 100% |
| Overall | 108.81% | 83.68% | 104.70% | 102.81% | 100% |

Looking at table 6.6 we see that for the -1 day category, *Longest throughput time (LTPT)* performs substantially better than the other methods. For the other two categories however, *Highest order level (HOL)* is far superior to the others. Furthermore, Looking at the overall performance we seen that HOL really stands out as the best option, requiring only 83.68% of weighted increased machine capacity (compared to the average). We therefore

conclude that the "Highest order level" should be used to select the order to adjust at each step of the iterative approach. Only if you happen to be dealing with a case in which you are specifically looking to expedite an order by a single day from its otherwise earliest possible due date, do we recommend using the longest throughput time method.

**Step size**

The procedure invoked for must-have orders decreases the processing time of an operation in steps. In each step the processing time is decreased by a certain amount of time, e.g., five minutes. It does so in an effort to find a minimal decrease in processing time which will allow the operation to be expedited, in hopes of making the order fit before a certain date. As explained in Section 6.2.2, The size of this decrease affects both the accuracy of the solution as well as the number of steps needed. Decreasing the amount of time by which we decrease the processing time each step, makes the result more accurate but increases the number of steps it takes to get there. More steps means more runs/ iterations of the algorithm, which in turn means more computational effort and a longer running time. In this way we are essentially dealing with a trade-off between computation time and accuracy.

All things considered we know that we want to work with the smallest possible step size that is still feasible. Which in our case would mean that the entire order acceptance procedure can still be run in at most roughly 2 minutes, for the biggest customers. To this extend we have experimented with various possible time step sizes. Table 6.7 shows the results of these experiments for different amounts of must-have orders evaluated at once (2, 3, and 5). Figure 6.2 graphs these same results. We see that with a step size smaller

TABLE 6.7: Step size experiment results (computation time in seconds)

| Step size | 2 orders | 3 orders | 5 orders |
|---|---|---|---|
| 1 hour | 0.98 | 1.14 | 1.23 |
| 30 min. | 1.31 | 1.27 | 1.68 |
| 15 min. | 1.75 | 1.92 | 2.36 |
| 10 min. | 2.47 | 2.58 | 3.31 |
| 5 min. | 5.20 | 5.79 | 9.32 |
| 2 min. | 24.74 | 27.03 | 46.79 |
| 1 min. | 83.92 | 88.12 | 167.83 |

than 5 minutes (2 or 1) the computation time starts to explode. Furthermore, despite wanting to use as small as step as possible, we have no need to go lower than 5 minutes. This is because the schedules in Limis Planner will not be executed to the exact minute anyway. Even more so, the scheduler adds 5 minutes to every operation to account for stochasticity anyway. Therefore, we recommend a step size of 5 minutes, as the resulting computation time is still more than feasible and there is no real need to be more precise than that anyway. Only in cases of a really large amount of must-have orders that need to be evaluated at once (and actually require adjustment) could you consider using 10 minutes instead of 5.

### 6.3.3 Classification model experiment results

In this subsection we present the results of the experiments with classification models to predict whether an order fits before a requested date. We used R to perform logistics regression, fit a decision tree model, and fit support vector machines with three different

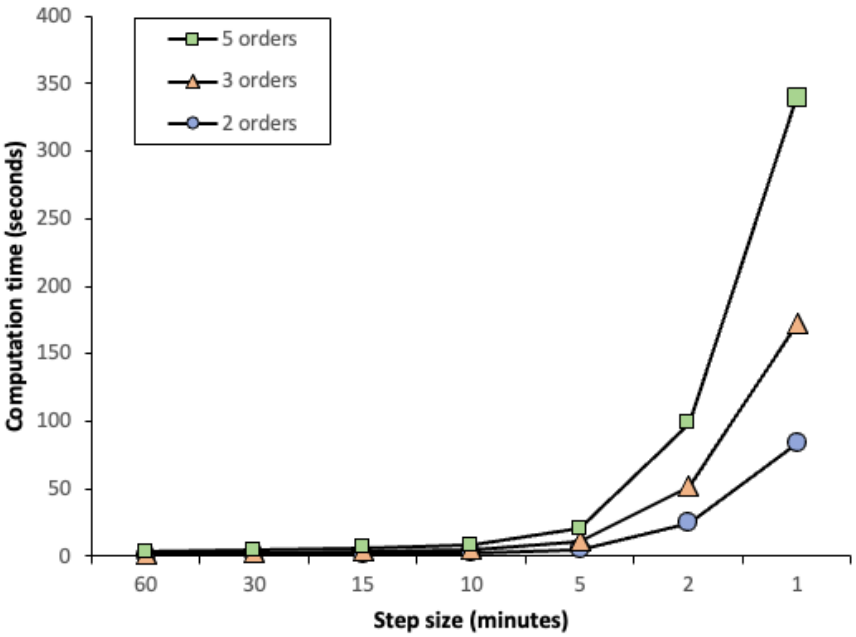



FIGURE 6.2: Plotted results of the step size experiments

types of kernels. Table 6.8 shows the results of the experiments (model performance) using an 80/20 train-test split.

These results show that the decision tree model performs by far the best, i.e., it has the best score in each category by a decent margin. The only exception being the sensitivity in which two others also score the full 100%. Support vector machine with radial kernel performs the second best, the biggest difference compared to decision tree being the precision. The precision is the area where all the models perform the worst, even going as shockingly low as 16.67% for the polynomial kernel. This is a bit of an issue, as precision is important when false positives are of higher concern than false negatives, which is certainly the case here. Accepting an order only to find out that it was actually not possible would seemingly be much more of a burden than quoting a later due date because you incorrectly thought it would not be possible.

TABLE 6.8: Classification model results

| Model | Accuracy | Sensitivity | Specificity | Precision |
|---|---|---|---|---|
| Decision tree | **95.45%** | **100%** | **93.75%** | **85.71%** |
| Logistics regression | 59.09% | 83.33% | 50% | 38.56% |
| Support vector machine: | | | | |
| *Linear* | 81.81% | 66.67% | 87.50% | 66.67% |
| *Polynomial* | 77.27% | **100%** | 76.19% | 16.67% |
| *Radial* | 90.90% | **100%** | 88.88% | 66.66% |

Figure 6.3 shows the generated tree. In this case it uses 2 decisions. First it looks at the number of days left until the due date - the length ideal critical path. If this is greater than, or equal to 5 it classifies it as fitting. If it is less than 5 it looks at the average daily

available capacity, if this is less than 11 hours it classifies the case as not fitting, else as fitting.

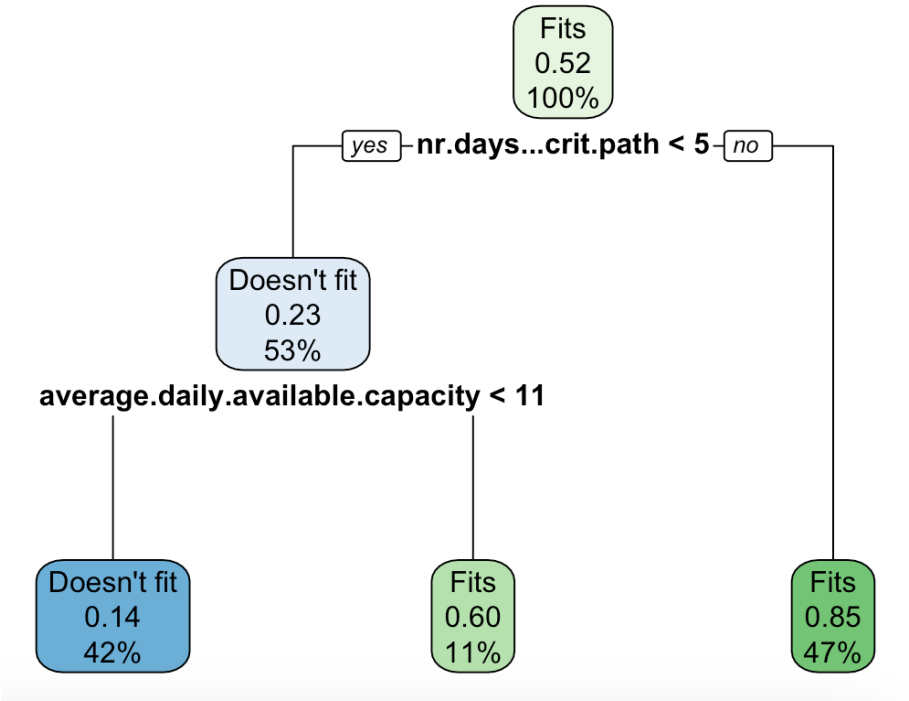



FIGURE 6.3: Decision tree generated by rpart.

Especially because we have only a limited amount of data, we can not just take the results based on the train test for granted. We therefore also perform 5-fold cross validation. (k-fold cross validation with k = 5). K-fold cross validation is a resampling method where K refers to the number of groups that data samples are split into. According to Kuhn Johnson (2013) one typically performs k-fold cross validations with K = 5 or k = 10, as those values have empirically shown to result in test error rates that don't have too much variance nor have too much bias. Table 6.9 shows the model performance using 5-fold cross validation.

TABLE 6.9: Classification model 5-fold cross validation results

| Model | Accuracy |
|---|---|
| Decision tree | 77.27% |
| Logistics regression | 71.83% |
| Support vector machine | |
| *Linear* | 69.95% |
| *Polynomial* | 76.38% |
| *Radial* | 77.05% |

We now see drastically lower levels of performance, i.e., the highest accuracy is now 77,27% instead of 95.45%. Furthermore, the performance of the different models are much closer to each other, all ranging between roughly 70 and 77%. It is notable that the decision tree model still performs the best.

Where 95% accuracy looked very promising, 77% is much less exciting. That means that almost 1 in 4 cases are judged incorrectly. This would simply be a far too large error rate for any practical use. However, we were dealing with, for machine learning standards, very little data, due to the limited amount of time available to gather data. This here only served as a first exploration into the possible use of classification models in order acceptance. The fact that we were able to attain a 95.45% accuracy in the train/ test split, does show that there might be potential. It would be interesting to see the level of performance that is achievable if the models are provided with much more data to train on.

Despite the decent accuracy results, the relatively low precision remains a concern. Therefore, we can not at this time recommend using / and or investing in the use of classification models for the purpose of order acceptance. Only if Limis, at some future point of time, really has spare time and/or resources would we suggest looking further into the possibility of using classification models in this manner.

## 6.4 Conclusion

In conclusion, the performance and comparison analysis of the proposed order acceptance tool reveal significant advantages over the current order acceptance tool.

In regards to computation time, it is more than clear that the proposed tool operates at a scale many orders of magnitude faster than the old method. Particularly for larger customers, the difference in computation time is substantial, reducing the time required from several hours to mere seconds. The new tool effectively only scales with the specifics of the order under evaluation and the volume of orders to be evaluated, offering a substantial boost in computational efficiency compared to the old solution.

Once fully developed and integrated within Limis Planner the proposed tool will be able to make accurate order acceptance decisions in a mere fraction of the time required by the current tool. The successful development of this tool is poised to have a transformative impact on order acceptance procedures within Limis customers, enabling these organization to make timely, informed order acceptance decisions.

Furthermore we have found that generally the procedure should be ran with step size of 5 minutes, and should use the Highest Order Level (HOL) method to select the next operation to adjust. Only when you explicitly know that you want to expedite the order by a single day, should you consider using the Longest Troughput (LTPT) method.

Finally, the experiments on classification models have shown that at this time, at least with our amount of training data available, they are not sufficiently capable of determining whether or not an order fits before a requested due date. They are not accurate enough and especially the precision of the tested models seems to form an issue. However, we've also seen some positive results, and can therefore not say that they could not potentially work better given more/ better training data.

Once the classification models have been trained, they can evaluate and find answers to cases practically instantaneously, even faster than the proposed finite loading approach. However, there is no conceivable practical scenario in which the proposed method would

not be more than quick enough. Furthermore, we strongly believe that once the finite loading method is capable of working with individual machine capacities and has been fully integrated, it should be more than accurate enough for Limis customers' business needs. Although we can not definitely rule it out, based on what we have seen, we do not believe that the classification model method could ever attain such accuracy for generalized cases.

All in all, we therefore would not recommend Limis to spend time and resources on this at the current moment of time. However, it could be worth revisiting in the future if they have spare time and/or resources to do so.

# Chapter 7

# Discussion

This chapter is dedicated to answering the research questions and giving a concise answer to the research goal. The chapter presents the research limitations, recommendations for the organization, as well as discussing the contribution to scientific literature and the possibilities for future research.

## 7.1 Conclusion

This research was originally motivated by two things. The first was the desire for Limis to explore the possibilities of AI and reinforcement learning based methods within their advanced planning and scheduling software. The second, and really the main motivation, were the challenges faced by Limis' customers in the realm of order acceptance. Many of these customers, as well as many other manufacturers, regularly struggle with questions such as *"can I deliver this order by this requested due date?"* or *What is a feasible due date that I can quote my customer for this order?*. Limis has previously made an effort to help their customers with these problems by developing their order acceptance tool. A big drawback however, is that this tool simply takes too long to come up with an answer. This is especially a problem for their bigger clients, as for them it may take up to several hours to find an answer. This because it requires a full on run of the scheduler to find a solution. This lead us to the following main research goal:

**How can a quick order acceptance tool be designed within Limis Planner in order to respond to, and to arriving customer orders?**

We first analyzed the context of the situation at hand within Limis, their existing software (Limis Start), and their customers. In designing a suitable solution to the order acceptance problem, we had to take into account that their customers typically work with orders with complex assembly-type routing, various machine groups with multiple (identical), and possibly even external, resources, with varying day specific operating hours. Furthermore, we learned that the scheduler generates schedules based on dispatching rules, which use release and due date derived from an infinite backward loading procedure.

Recognizing both the challenges and opportunities that lay ahead, we turned to extensive literature review to find a basis for a possible solution. In doing so we delved deeper into order acceptance and related topics such as due date assignment, rescheduling, and capacity control methods. We also looked into AI based solutions but quickly came to the realization that these were really not practical, nor relevant for us, considering that we determined scheduling to be outside of the scope. This because there seems to be no relevant

research on AI based order acceptance, at least not in the way we view order acceptance. The only literature we found on this was purely concerned with accepting an order or reserving capacity for a potentially more profitable order, which we also determined to be outside of the scope. We did benefit from literature, by finding a basis for our solution in capacity loading procedures that essential combine order acceptance, due date assignment, and capacity control.

Due to the desire for a quick tool, we decided to use finite loading method in combination with an iterative approach that adjusts the processing time of operations, to mimic increased machine capacity. This method allows us to observe currently planned capacity and use the left over available capacity to load new order on top of the existing schedule without altering it. A big benefit of this, as the experiment results would later reflect, is that we save a lot of time not having to deal with (re)scheduling the existing orders.

The proposed method essentially consists of three parts and two phases. In the first phase we check if the order fits before the requested due date, meaning that each of its operations can be scheduled between the current date and its requested due date. If the order fits, we have our answer and the procedure stops. If the order does not fit, we move on to phase two, which depends on whether the order is a so called "must-have" order or a "best-fit" order. The distinction between the two come down to whether we believe it to be worth increasing capacity in order to meet the requested due date for the order or not. If this is the case, we come to the iterative approach that tries to find the required capacity adjustment to make the order fit, by repeatedly increasing machine capacity levels and rerunning the backward finite loading procedure. If we do not want to increase capacity, we instead come to the due date assignment part of the solution, in which we use finite forward loading to find the first suitable alternative due date.

Experiments have shown that for the capacity control part it is best to use a step size of 5 minutes and to use a operation selecting rule based on the highest order level (HOL) to select an operation to adjust. Most importantly, the performance analysis has shed light on the performance of the proposed method and tool in comparison to the existing one. When it comes to computation time, the superiority of the proposed method is clear. It operates several orders of magnitude faster than the existing method, marking a substantial reduction in processing time, particularly for larger customers. This great efficiency improvement ensures that the newly proposed tool can handle a wide range of order complexities and volumes, offering a more than substantial boost in usability, and ultimately streamlining the entire order acceptance process.

We were unable to work with machine capacities on an individual machine level. Instead, we had to settle for working with machine group capacities. As a result, we concluded that, at this point in time, there is no purpose in comparing the solution quality (accuracy) of the proposed method to that of the scheduler and the existing method. This because as a result of working with machine groups, the new method consistently find much earlier dates than the existing method.

Instead of selectively choosing cases where the proposed method consistently aligns with the scheduler, such as using past or exceedingly far out due dates, risking the study's integrity, we chose to refrain from further evaluating solution quality within the scope of this study. We strongly urge Limis to pick this up and conduct this assessment themselves,

after adjusting Limis Planner to record machine capacities on an individual machine level and adjusting the proposed method accordingly.

We have however, readily shown that the proposed method returns valid results, i.e., every operation of an order is correctly loaded on top of existing schedule. That is assuming the situation in which machine capacities of individual machines of the same type and characteristics (belonging to the same machine group) can indeed be perfectly pooled between them. This means that we have, in so far, succeeded in creating a quick tool, that is able to solve the order acceptance problem based on capacity left available as a result of existing schedules. Furthermore, we believe that when fully developed and integrated with the scheduler it could completely replace the existing tool, rather than being used along side it / being an extension to. The speed at which it completes its calculations, makes the tool a suitable and practical solution for operational purposes, ensuring that businesses can make timely and informed order acceptance decisions with confidence.

The experiments on classification models have shown that at this time, at least with our amount of training data available, they are not sufficiently capable of determining whether or not an order fits before a requested due date. They are not accurate enough and especially the precision of the tested models seems to form an issue. However, we've also seen some positive results, and can therefore not say that they could not potentially work better given more/ better training data.

Once the classification models have been trained, they can evaluate cases practically instantaneously, even faster than the proposed finite loading approach. However, there is no conceivable practical scenario in which the proposed method would not be more than quick enough. Furthermore, we strongly believe that once the finite loading method is capable of working with individual machine capacities and has been fully integrated, it should be more than accurate enough for Limis customers' business needs. Although we can not definitely rule it out, based on what we have seen, we do not believe that the classification model method could ever attain such accuracy for generalized cases.

Next to designing the proposed method, we have also already developed the actual tool that can be used to perform the proposed methods on real customer data, directly from their database, in real time. Even more so it has already largely been integrated within Limis Planner, to work together with the interface of the current order acceptance tool. It is important to note that the use of the new proposed tool and the existing tool are not mutually exclusive. In fact they could very well be used together, the new tool to quickly asses whether an order fits, and if not where we probably need extra capacity, or alternatively find the earliest feasible due date. This allows Limis customers to quickly relay this to their respective customers. The old tool can then be run using the results of the new tool as input, to be even more certain of where the new order will actually end up in the new schedule, and what this means for its expected due date.

Finally, the code of which the tool (and method) is comprised can be re-used by Limis for other projects and applications as they see fit. In fact the code that loads, transforms and sets up all the machine and order data is already being used in a separate project and possibly a different study with regards to AI-based scheduling within Limis, Fraunhofer and the University of Twente.

## 7.2 Limitations

A notable limitation is that the proposed method, unlike the old method is not based on the detailed scheduler within Limis Planner. While this allows it to be many orders of magnitude faster, it also means that the results may not perfectly align with the results of the scheduler. This because, amongst other things, detailed scheduling in Limis Planner is based on dispatching rules, and the addition of the new order may therefore alter the existing schedule, while our proposed method does not. This potentially leads to some over estimation of due dates, which is presumed to be less of an issue than underestimating them. Furthermore, the old acceptance tool can still be run on top of the new tool, in situations in which you might really want to know the exact place of the new order in the new schedule.

Another limitation is that at least for now the solution is purely based on machine capacity and does not include man capacity. These are often very similar however, especially for smaller customers there might be some observable differences. We didn't think this to be very relevant in showing the effectiveness of the proposed method, and was therefore not included. It should however not require too much work to include this in a future version.

Finally, another limitation is that in the current version of the proposed tool we only work with lead times of required materials. This means that we do not work with inventory levels over time, but rather only check if a planned date is further out than the lead time of its required materials. In doing so we essentially assume that we hold no stock for these materials. While this is a limitation compared to reality, it is an improvement over existing methods in literature, as these (typically) do not work with/ take into account materials at all.

## 7.3 Recommendations

First of all, we recommend not wasting resources on developing an AI / reinforcement learning based approach to order acceptance, at this time. Secondly, we strongly recommend finishing and implementing the proposed solution and its resulting tool. To this end we above all urge Limis to start recording machine capacity on individual machine level, to adjust the developed order acceptance tool accordingly and to then conduct thorough experimentation on the accuracy of the tools found results.

In its current form the tool itself would require little effort to be made ready for a beta release to interested customers. After this we recommend testing the tool in practice at an actual customer, i.e., having the customer test it for a while and give their feedback, both on the used of the tool as well as in regards to how well the tool's found results match with reality.

We believe that, while it in the future could, the proposed tool does not necessarily have to completely replace the existing tool, but rather could function along side it. The new tool could be used by customers to find quick answers, while the old tool could be used afterwards to confirm the found results by the new tool. Furthermore, we suggest further developing/ extending the proposed tool to include even more practical features. We believe that there is enough desire amongst existing and potential customers for such a

tool to warrant further investment in development. Having showcased/ demonstrated the tool to an actual customer, and considering their very positive feedback, we feel even more certain about the benefits of the proposed solution.

## 7.4 Scientific contribution

Our first scientific contribution is that we essentially combine an order acceptance problem, a due date assignment problem, and a capacity problem into one propose a solution to these based on finite capacity loading. Furthermore, we extend this with practical aspects such as orders with complex assembly-type routing, machine groups consisting of multiple (identical) resources, with varying date specific operating hours. Even more so, we extend on literature by taking into account material requirements, by confirming that planned operations do not violate their material lead times. We notably also combine the capacity loading method with an iterative approach to find a possible machine capacity adjustment that allows the expediting of an order, to make it fit before a requested due date. Also, we've demonstrated that it is possible to to bring such a method to practice in a dynamic, scale-able approach that works independently of characteristics of the manufacturing environment. Finally, we've explored the use of classification models for internal order acceptance decisions, i.e., classifying an order as fitting or not fitting before a certain date, given 17 different input parameters. The use of classification models in this exact manner seems to be a novel concept.

## 7.5 Future research

There are various possibilities for future research. One could research predictive models that anticipate changes in order requirements and resource availability, enabling proactive scheduling and possibly even better decision making. Another possible interesting topic might be supply chain integration, i.e., investigating the integration of these order acceptance methods within the broader supply chain. This could enable more transparency between parties in a supply chain and allow better coordination and improved production planning. Something more closely related, that might be interesting as a follow up, is to research how the proposed methods would perform in practice. So implementing the methods within an actual company and observing over time how the tools predictions match up against the actual deliver dates in reality. Another interesting idea might be to research the addition of more traditional order acceptance problems, i.e., do we want to reserve capacity for potential orders arriving in the future. You might want to do this either to leave room for potentially more profitable orders and/or more important orders. This might give us insight into the trade-off between delaying certain orders, to be able to expedite others and the resulting reward and/or penalty.

# Bibliography


[1] M.Emin Aydin and Ercan Öztemel. Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems*, 33(2):169–178, 2000. URL: https://www.sciencedirect.com/science/article/pii/S0921889000000877, doi:https://doi.org/10.1016/S0921-8890(00)00087-7.

[2] J. Bertrand. The effect of workload dependent due-dates on job shop performance. *Management Science*, 29:799–816, 07 1983. doi:10.1287/mnsc.29.7.799.

[3] J. Bertrand. The use of workload information to control job lateness in controlled and uncontrolled release production systems. *Journal of Operations Management*, 3:79–92, 02 1983. doi:10.1016/0272-6963(83)90009-8.

[4] C. Duron, M.A. Ould Louly, and J.-M. Proth. The one machine scheduling problem: Insertion of a job under the real-time constraint. *European Journal of Operational Research*, 199(3):695–701, 2009. URL: https://www.sciencedirect.com/science/article/pii/S037722170800458X, doi:https://doi.org/10.1016/j.ejor.2007.09.048.

[5] Yong Gui, Dunbing Tang, Haihua Zhu, Yi Zhang, and Zequn Zhang. Dynamic scheduling for flexible job shop using a deep reinforcement learning approach. *Computers & Industrial Engineering*, 180:109255, 2023.

[6] Hans Heerkens and Arnold van Winden. *Systematisch managementproblemen oplossen*. Noordhoff Uitgevers, 2017.

[7] Johannes M.G. Heerkens and A. van Winden. *Geen probleem, een aanpak voor alle bedrijfskundige vragen en mysteries*. Business School Nederland, 2012. Boekredactie.

[8] Jiahuan Huang, Aimin Wang, and Tao Ding. Progressive order insertion dynamic scheduling method considering production capacity. In *2022 3rd International Conference on Computer Science and Management Technology (ICCSMT)*, pages 512–515, 2022. doi:10.1109/ICCSMT58129.2022.00114.

[9] Jiahuan Huang, Aimin Wang, and Tao Ding. Progressive order insertion dynamic scheduling method considering production capacity. In *2022 3rd International Conference on Computer Science and Management Technology (ICCSMT)*, pages 512–515, 2022. doi:10.1109/ICCSMT58129.2022.00114.

[10] Pinar Keskinocak and Sridhar Tayur. *Due Date Management Policies*, pages 485–554. Springer US, Boston, MA, 2004. doi:10.1007/978-1-4020-7953-5_12.

[11] B.G. Kingsman, I.P. Tatsiopoulos, and L.C. Hendry. A structural methodology for managing manufacturing lead times in make-to-order companies. *European Journal of Operational Research*, 40(2):196–209, 1989. URL: https://www.

sciencedirect.com/science/article/pii/0377221789903305, doi:https://doi.org/10.1016/0377-2217(89)90330-5.

[12] Patrick Moratori, Sanja Petrovic, and Antonio Vázquez. Match-up strategies for job shop rescheduling. In Ngoc Thanh Nguyen, Leszek Borzemski, Adam Grzech, and Moonis Ali, editors, *New Frontiers in Applied Artificial Intelligence*, pages 119–128, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[13] Joel Oren, Chana Ross, Maksym Lefarov, Felix Richter, Ayal Taitler, Zohar Feldman, Christian Daniel, and Dotan Di Castro. SOLO: search online, learn offline for combinatorial optimization problems. *CoRR*, abs/2104.01646, 2021. URL: https://arxiv.org/abs/2104.01646, arXiv:2104.01646.

[14] Jorge Palombarini and Ernesto Martinez. Smartgantt–an interactive system for generating and updating rescheduling knowledge using relational abstractions. *Computers & chemical engineering*, 47:202–216, 2012.

[15] Jorge A Palombarini and Ernesto C Martínez. Closed-loop rescheduling using deep reinforcement learning. *IFAC-PapersOnLine*, 52(1):231–236, 2019.

[16] Kelli Robinson and Scott Moses. Effect of granularity of resource availability on the accuracy of due date assignment. *International Journal of Production Research - INT J PROD RES*, 44:5391–5414, 12 2006. doi:10.1080/00207540600665810.

[17] Manuel Schneckenreither, Sebastian Windmueller, and Stefan Haeussler. Smart short term capacity planning: a reinforcement learning approach. In *Advances in Production Management Systems. Artificial Intelligence for Sustainable and Resilient Production Systems: IFIP WG 5.7 International Conference, APMS 2021, Nantes, France, September 5–9, 2021, Proceedings, Part I*, pages 258–266. Springer, 2021.

[18] Susan A. Slotnick. Order acceptance and scheduling: A taxonomy and review. *European Journal of Operational Research*, 212(1):1–11, 2011. URL: https://www.sciencedirect.com/science/article/pii/S0377221710006405, doi:https://doi.org/10.1016/j.ejor.2010.09.042.

[19] M. Thurer, M. Stevenson, C. Silva, M.J. Land, and M. Filho. Workload control and order release in two-level multi-stage job shops: An assessment by simulation. *International Journal of Production Research*, 51(3):869–882, February 2013. doi:10.1080/00207543.2012.676685.

[20] Guilherme Vieira, Jeffrey Herrmann, and Edward Lin. Rescheduling manufacturing systems: A framework of strategies, policies, and methods. *J. Scheduling*, 6:39–62, 01 2003. doi:10.1023/A:1022235519958.

[21] Hao Wang, Junfu Cheng, Chang Liu, Yuanyuan Zhang, Shunfang Hu, and Liangyin Chen. Multi-objective reinforcement learning framework for dynamic flexible job shop scheduling problem with uncertain events. *Applied Soft Computing*, 131:109717, 2022.

[22] Yu-Fang Wang. Adaptive job shop scheduling strategy based on weighted q-learning algorithm. *Journal of Intelligent Manufacturing*, 31(2):417–432, 2020.

[23] Yi Zhang, Haihua Zhu, Dunbing Tang, Tong Zhou, and Yong Gui. Dynamic job shop scheduling based on deep reinforcement learning for multi-agent manufacturing systems. *Robotics and Computer-Integrated Manufacturing*,

78:102412, 2022. URL: https://www.sciencedirect.com/science/article/pii/S0736584522000977, doi:https://doi.org/10.1016/j.rcim.2022.102412.

[24] Yejian Zhao, Yanhong Wang, Yuanyuan Tan, Jun Zhang, and Hongxia Yu. Dynamic jobshop scheduling algorithm based on deep q network. *IEEE Access*, 9:122995–123011, 2021. doi:10.1109/ACCESS.2021.3110242.

# Appendix A

# Variable table

The table can be found on the next page.

| group | Variable | Type | Used in function | Taken from | DataBase variable | Used for |
|---|---|---|---|---|---|---|
| **werkplek / machine** | machineID | string | InitFactory() | X | **X** | machine type |
| | extCheck | boolean | InitFactory() | X | **X** | is het een externe werkplek ja of nee |
| | verzamelCheck | boolean | InitFactory() | X | **X** | is het een verzamel werkplek ja of nee |
| | Capacity | double | InitializeCapacityInfoForYears() | X | **X** | maximaal aantal uren op deze werkplek (op deze dag) |
| | PlannedCapacity | double | LoadOrders() | X | **X** | reeds ingeplande aantal uren op deze werkplek (op deze dag) |
| | available capacity | double | InitializeCapacityInfoForYears() | - | **-** | Nog beschikbaar aantal uren op deze werkplek (op deze dag) |
| | doorlooptijd | double | | | | doorloop tijd van bewerking indien externe werkplek (in dagen) |
| | | | | | | |
| **project** | ProjectNumber | string | LoadOrders() | X | **X** | project id |
| | DueDate | dateTime | LoadOrders() | X | **X** | gevraagde leverdatum |
| | what if | boolean | LoadOrders() | X | **X** | is de order een what_if order |
| | best fit | boolean | - | - | **-** | is de order een best-fit of een must-have order |
| | orders | list | LoadOrders() | - | **-** | lijst met orders die bij het project horen |
| | | | | | | |
| **orders** | ProjectNumber | string | LoadOrders() | X | **X** | project waar de order onder valt |
| | OrderNumber | string | LoadOrders() | X | **X** | order id |
| | bewerkingen | list | LoadOrders() | - | **-** | lijst met bewerkingen die bij de order horen |
| | | | | | | |
| **bewerkingen** | ProjectNumber | string | LoadOrders() | X | **X** | project waar de bewerking onder valt |
| | OrderNumber | string | LoadOrders() | X | **X** | order waar de bewerking onder valt |
| | stapNumber | string | LoadOrders() | X | **X** | bewerking id |
| | machine | string | LoadOrders() | X | **X** | machine waarop de bewerking gedaan moet worden |
| | processing time | double | LoadOrders() | X | **X** | processing time van bewerking |
| | wachttijd | double | LoadOrders() | X | **X** | wacht / droog tijd |
| | volgende bewerking | string | LoadOrders() | X | **X** | stap_nr van volgende bewerking (eigen als geen volgende) |
| | materialen | list | LoadOrders() | X | **X** | lijst met materialen benodigd voor bewerkingen |
| | doorlooptijd | double | LoadOrders() | X | **X** | doorloop tijd van bewerking |
| | Max uren per dag | double | SetCapDays() | X | X | het maximaal aantal uren per dag, voor bewerkingen die langer dan 1 dag duren. |
| | | | | | | |
| **Materialen** | materiaalID | string | LoadOrders() | X | **X** | materiaal id |
| | lead time | int | LoadOrders() | X | **X** | lead time van materiaal (in dagen) |

# Appendix B

# Code snippets from Chapter 5

The following code snippet (snippet 1) shows both the checks on stopping criteria as well as the function for finding the longest lead time.

```
//difference between the day we're checking and now (will be
    compared to leadtimes)
TimeSpan difference = operation.StartDate.AddDays(-i) - DateTime.
    Now;
int daysAhead = (int)difference.TotalDays;
int requiredLeadTime = operation.longestLeadTime();
//check if checking date is further out than now
if (operation.StartDate.AddDays(-i) > DateTime.Now.AddDays(
    startTomorow -1))
{
    //check if the checking date is further out than the longest
    required lead time
    if (daysAhead >= requiredLeadTime)
    {
    ...
    }
...
}

public int longestLeadTime()
{
    int llt = 0;
    foreach (var mat in Materials)
    {
        if (mat.LeadTime > llt)
        {
            llt = mat.LeadTime;
        }
    }
    return llt;
}
```

```
public bool MultiDayCheck(Operation op, Machine machine, DateTime
    date)
{
    double totalCapacity = 0;
    int closedDays = Factory.Instance.NrClosedDays(date, date.
    AddDays(op.ExpectedDays + ((op.ExpectedDays / 5) * 2)), machine)
```

```
    ;
    for (int i = 0; i < op.ThroughputTime + closedDays; i++)
    {
        //Check if not the first day, if not a closed day,
        //and check if available capacity is a full single machine
    days worth (no interuptions)
        if (i > 0 && machine.CapacityData[date.AddDays(i).Date].
    Capacity > 0 &&
            machine.CapacityData[date.AddDays(i).Date].
    AvailableCapacity <
            machine.DayCap[date.AddDays(i).DayOfWeek] &&
    totalCapacity < op.ProcessingTime)
        {
            return false;
        }
        totalCapacity += Math.Min(machine.DayCap[date.AddDays(i).
    DayOfWeek],
            Math.Max(0, machine.CapacityData[date.AddDays(i).Date].
    AvailableCapacity));
    }
    if (totalCapacity > op.ProcessingTime)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

The above code snippet (snippet 2) shows the so called multi-day checking procedure, that does exactly that. We see that the procedure checks if the days after the first day have at least a full day available until the total capacity is greater than the operation processing time. Note that it takes into account closed days and correctly adjusts for them. Else it would incorrectly reject the operation if a closed day (weekend, holiday, etc.) during the period it is checking. In fact an order that takes longer than 5 days would become impossible to fit, assuming we're dealing with a machine that is closed over the weekend.

The following code snippet (snippet 3) shows the procedure used to find the critical path length of an order. This was again meticulously tested to function correctly.

```
public int FindCriticalPathLength(Project project)
{
    List<int> ThroughputTimes = new List<int>();
    foreach (Order order in project.Orders.Values)
    {
        order.InclusiveThroughputTime =  CalcMaxThroughput(order);
        ThroughputTimes.Add(order.InclusiveThroughputTime);
    }
    if (ThroughputTimes == null || ThroughputTimes.Count == 0)
    {
        return 0;
    }
    else
    {
```

```
        return ThroughputTimes.Max();
    }
}

public int CalcMaxThroughput(Order order)
{
    if (order.SubOrders == null || order.SubOrders.Count == 0)
    {
        order.InclusiveThroughputTime = order.OrderThroughputTime()
    ;
        return order.OrderThroughputTime();
    }
    else
    {
        List<int> ThroughputTimes = new List<int>();

        foreach (Order subOrder in order.SubOrders.Values)
        {
            ThroughputTimes.Add(CalcMaxThroughput(subOrder));
        }
        order.InclusiveThroughputTime = order.OrderThroughputTime()
    + ThroughputTimes.Max();
        return order.OrderThroughputTime() + ThroughputTimes.Max();
    }
}
```

The following code snippet (snippet 4) shows the procedure that we have written, that takes care of this for us.

```
public List<Operation> GetAllOperationsFromOrders(Order order)
{
    List<Operation> allOperations = new List<Operation>();

    // Include operations from the current order
    allOperations.AddRange(order.Operations.Where(operation => !
    operation.alreadyReduced));

    // Recursively include operations from suborders
    foreach (var subOrder in order.SubOrders)
    {
        allOperations.AddRange(GetAllOperationsFromOrders(subOrder.
    Value));
    }

    return allOperations;
}

public Operation LongestThroughputTime (DateTime duedate)
{
    List<Operation> allOperations = Orders.Values.SelectMany(order
    => OrderManager.Instance.GetAllOperationsFromOrders(order)).
    ToList();
    Operation longestOp = allOperations.FirstOrDefault(); ;
    TimeSpan longestTime = TimeSpan.Zero;
    TimeSpan throughputTime = TimeSpan.Zero;
    if (allOperations.Count > 1)
```

```
    {
        for (int i = 0; i < allOperations.Count; i++)
        {
            var curOp = allOperations[i];
            if (i < allOperations.Count - 1)
            {
                var nextOp = allOperations[i + 1];
                throughputTime = (nextOp.PlannedDate - curOp.
PlannedDate).Duration() - TimeSpan.FromDays(
                    Factory.Instance.NrClosedDays(nextOp.
PlannedDate, curOp.PlannedDate,
                    Factory.Instance.MachinesByType[curOp.Machine].
First()) + curOp.ThroughputTime);
            }
            else
            {
                throughputTime = (curOp.PlannedDate - duedate).
Duration() - TimeSpan.FromDays(
                    Factory.Instance.NrClosedDays(curOp.PlannedDate
, duedate,
                    Factory.Instance.MachinesByType[curOp.Machine].
First()) + curOp.ThroughputTime);
            }
            //compare the throughputTime to the longest so far
            if (throughputTime > longestTime)
            {
                longestTime = throughputTime;
                longestOp = curOp;
            }
        }
    }
    return longestOp;
}
```

After we've found the operation with the longest throughput time, we want to decrease it's processing time before running the backward loading procedure again. Furthermore, we need to keep track of the amount of time we reduce the order by and on which machine and date we end up doing this as show in the code below. (snippet 5)

```
//Find the operation with the longest throughput
Operation longestOp = project.LongestThroughputTime(duedate);
//reduce the processing time by the value of the textbox
longestOp.ProcessingTime -= reductionHours;
// Write down the reduction
longestOp.Reduction += reductionHours;
```

Finally, we need to go over all the reductions that we have found and add them to the reductions table so that they are correctly displayed in the tool. After adding the reduction to the project, we set the processing time of the operation back to its original, and its reduction back to 0. We do this so that we can perform multiple runs without having to restart the tool. Without this, the reductions keep adding up and the processing times get all messed up. The used procedure can be seen in the code snippet below (snippet 6).

```
List<Operation> allOperations = project.Orders.Values
```

```
        .SelectMany(order => OrderManager.Instance.
    GetAllOperationsFromOrders2(order))
        .ToList();
foreach (var operation in allOperations)
{
    if (operation.Reduction > 0)
    {
        Reduction red = new Reduction
        {
            date = operation.PlannedDate,
            machineNr = operation.Machine,
            reduction = operation.Reduction,
            projectNr = operation.ProjectNumber + "." + operation.
    OrderNumber
        };
        project.Reductions.Add(red);
        operation.ProcessingTime = operation.OriginalProcessingTime
    ;
        operation.Reduction = 0;
    }
}
```